SCO
THE SANTA CRUZ OPERATION

SCO® Product Engineering Toolkit Guide

Product Engineering Toolkit Guide

Version 4.0.0

The Santa Cruz Operation, Inc.

Document version: 4.0.0C

Date: 25 June 1990

Contents

Chapter 1

Introduction

Introduction

The SCO Product Engineering Toolkit, hereafter called the "Toolkit," makes an application installable with the **custom** utility. Under SCO XENIX System V®, **custom** is described in the Commands (C) section; under SCO UNIX System V®/386 Release 3.2, **custom** is described in the System Administration (ADM) section of the respective reference manuals supplied with your system documentation.

When **custom** installs a software product, the files that comprise the product are transferred from the media on which the software product is distributed to the computer on which the product is being installed. Using the information supplied in a parameter list, **custom** places the files in the correct locations, assigns permissions, and performs optional tasks such as making device nodes.

This chapter covers the following topics:

- a description of this guide
- how to install the software supplied with the Toolkit

About This Guide

This guide is for developers who are preparing applications or device drivers for installation, or who are porting applications or device drivers from MS-DOS® or other forms of the UNIX Operating System to SCO UNIX System V or to SCO XENIX System V.

This guide assumes that you are familiar with the use of your operating system, and you are familiar with the principles of shell programming. In addition, you should be familiar with the documentation supplied with your operating system.

Procedures are discussed for preparing your application for installation using the utilities provided in the Toolkit. INSTALLATION means the entire process of transferring files from removable media to a target computer.

Contents of This Guide

Here is a brief description of each chapter in this guide:

- Chapter 1, this "Introduction," describes the major topics of the guide, product features, the conventions used in this book, and how to install the software supplied with your Toolkit.
- Chapter 2, "Cutting the Distribution," describes the Toolkit utilities and takes you through the procedures for using the **docut**(SCO), **mkcuts**(SCO), and **mkflops**(SCO) utilities.
- Chapter 3, "Installing an Application," presents the **custom** utility, and shows you how to install (as well as remove) your application.
- Chapter 4, "Writing Installation Scripts," describes special purpose scripts— for initialization, preparation, and removal—and the process for creating them.
- Chapter 5, "Installing Device Drivers," addresses the special characteristics in making device drivers installable, including guidelines, discussion of scripts, and compatibility with Semi-Automated Mass Installation (SAMI).
- Appendix A, "Tools and Information," contains manual pages for all of the utilities explained in this guide.

- Appendix B, "Customizing SCO Open Desktop or SCO Xsight Applications," describes some special features that these products offer.

- Appendix C, "Compiling Compatible Binaries," discusses the issues related to compiling binaries.

- A glossary and an index are provided at the end of this guide for your reference.

Required Software

The Toolkit contains all the software required to follow the directions and procedures explained in this guide. The Toolkit runs with all 386, or 486 versions of the SCO XENIX System V or SCO UNIX System V operating systems.

Before starting, on SCO UNIX System V systems, install the extended utilities so that you have access to the **man**(C) utility for viewing the manual pages supplied with the Toolkit.

On SCO XENIX System V systems, install the text processing package to get access to the **man**(C) utility. If this product is not available on your system, you can display the manual pages supplied with this product from the command line. For example, the following command can be used to view the **docut**(SCO) manual page:

```
pcat /usr/man/cat.SCO/docut.SCO | pg
```

These utilities are described on the **pcat**(C) and **pg**(C) manual pages in your *User's Reference* manual.

Required Hardware

The only hardware stipulation for executing the Toolkit is that your computer be equipped with an Intel 80386 or 80486 processor. The Toolkit is compatible with Microchannel, AT-bus, EISA bus, and C-bus equipped computers.

Product Features

The Toolkit provides several features that can be useful for optimizing how your distribution is produced. These features are:

- Compressing Files — You can create compressed distribution files so that you can conserve distribution media.
- Disk Image Copying — When making multiple copies of your software, you can create a single file for each distribution floppy disk to be copied. Because each file is the size of the target floppy disk, it can be copied readily to make the distribution copies.
- Packages — You can organize your product into units corresponding to pieces of your product, tasks, or use any size of your own choosing. When you create a file hierarchy of the files that you want installed, each unit becomes a directory and is called a package.
- Permissions List — The Toolkit automatically creates the permissions list for your distribution. This file instructs the **custom** utility during an installation to direct where files and directories are located, assigns permissions modes, and performs other tasks such as creating device nodes. This list, also called the "permlist," is used by system administration utilities.
- Sum List — You can create a sum list at the end of the distribution so that you can tell readily if any changes were made between copies of the distribution.

Examples, Symbols, and Guide Conventions

This guide uses several documentation conventions that match those used in the SCO UNIX System V and SCO XENIX System V documentation.

Screen Examples

Examples are shown in screen representations. Note that these are examples of the information that you will see on your screen. Some information, especially that seen when using the **custom** utility, can change depending on what other software products are installed on your system.

```
Here is an example of something seen on your screen
```

New Terms or Concepts

New terms or concepts, when introduced, are shown in all uppercase letters, LIKE THIS. These terms are listed and defined in the glossary.

Utility or Utility Options

Utility or utility options are shown in **boldface** type. For example, the **mkdev**(SCO) utility name is shown in boldface.

Manual Page References

References to manual pages for a specific utility are as follows: the utility name (shown in **bold**) is immediately followed by the abbreviated name of the section in the SCO UNIX System V and SCO XENIX System V reference manual containing that utility's manual page; the name of the reference section is enclosed by parentheses. For example, the manual page for the **tar** utility, located in the Commands (C) section of the *User's Reference* manual of the SCO UNIX System V and SCO XENIX System V documentation, is notated like this: **tar**(C).

Because this guide describes information that relates to both the SCO UNIX System V and SCO XENIX System V operating systems, manual page section references were selected to reflect that of the most current operating system. The following manual pages have section references that vary between these two operating systems:

Utility	SCO XENIX System V	SCO UNIX System V
configure	**configure**(C)	**configure**(ADM)
custom	**custom**(C)	**custom**(ADM)
fixperm	**fixperm**(M)	**fixperm**(ADM)

Arguments or User Supplied Values

An argument or value inside angle brackets (< >) is a value that you supply and is shown in *italics* font. The following comment line, for example, instructs you to enter the variable name and the value of that variable, separated by an equals sign (=):

#<*variablename*>=<*value*>

Pathnames

The actual filename of a script, such as *init.idd*, is shown in *italics*.

Pathnames of files, for example */tmp/perms*, are shown in *italics*.

Installing the Toolkit

Install the Toolkit software using the **custom** utility.

The following topics are discussed in this section:

- installing the SCO UNIX System V Toolkit
- installing the SCO XENIX System V Toolkit
- upgrading a SCO XENIX System V Toolkit
- international considerations

If you are upgrading an SCO XENIX System V version of the Toolkit from a previous release, examine the procedure for using SCO XENIX System V **custom**(M) described in this section and then read "Upgrading an SCO XENIX System V Toolkit" later in this section.

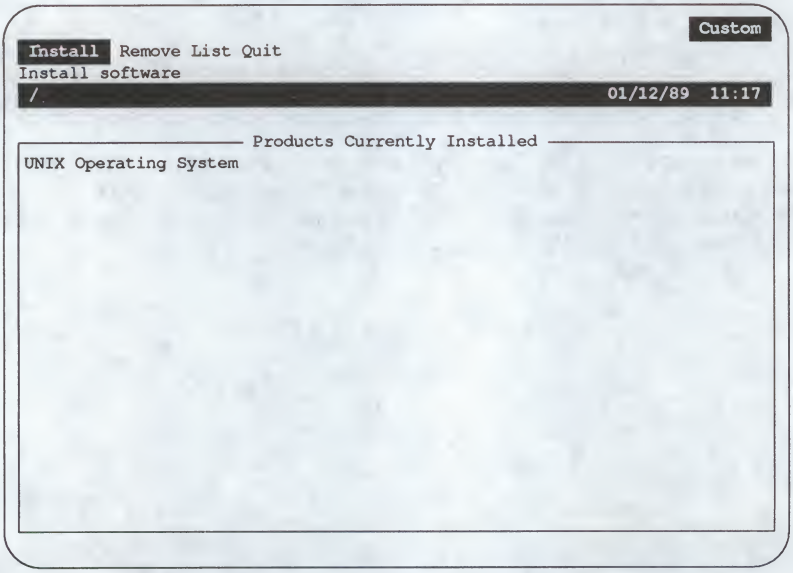
Installing the SCO UNIX System V Toolkit

Install the Toolkit under SCO UNIX System V **custom**(ADM) by first logging in as *root* and following this procedure:

1. Make the following **sysadmsh** selection:

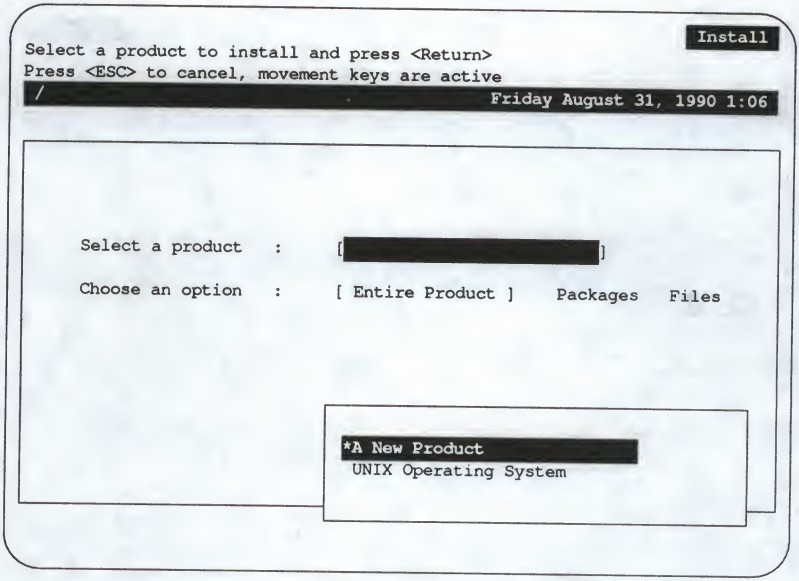
   System→Software

   The main **custom** menu is displayed:



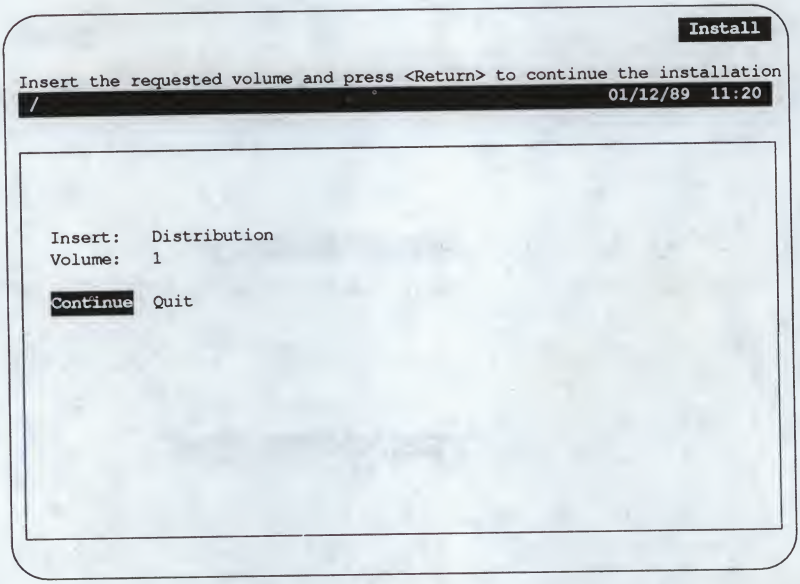
2. The menu is already set to select Install; press ⟨Return⟩ to continue.

You see the next screen:



3. A New Product is highlighted. Press ⟨Return⟩ and the Entire Product option is highlighted. Press ⟨Return⟩ once again.

The following menu is displayed:



4. Insert the first disk of your product distribution as instructed and press ⟨Return⟩. Shortly after you do so, you are prompted again to insert the first volume, this time by its actual name. Simply press ⟨Return⟩ and insert additional volumes as prompted.

5. When the product installation is complete, you are returned to the **custom** menu. If you are finished, exit **custom** by selecting Quit from the main **custom** menu and confirming your selection.

Installing the SCO XENIX System V Toolkit

Install the Toolkit by first logging in as *root* and following this procedure:

1. Enter **custom** and press ⟨Return⟩.

```
# custom
```

   The main **custom** menu is displayed:

```
1. Operating System
2. Development System
3. Text Processing System
4. Add a supported product

Select a set to customize or enter q to quit:
```

2. If you do not select a system to customize, enter **q** to stop the installation procedure. You need to invoke **custom** later to add other packages. To add the Toolkit, select option **4**.

   The following messages are displayed:

```
Installing custom data files ...

Insert the Toolkit floppy disk
and press <RETURN> or enter 'q' to quit:
```

3. Insert the diskette as instructed and press ⟨Return⟩.

The following submenu is displayed:

```
1.  Install one or more packages
2.  Remove one or more packages
3.  List the available packages
4.  List the files in a package
5.  Install a single file
6.  Select a new set to customize
7.  Display current disk usage
8.  Help

Select an option or enter q to quit:
```

4. Select option **1**. A similar list is displayed. Enter the name **petkit**. When you enter the name of a package, **custom** prompts for the necessary floppy disk. Insert the appropriate disk and follow the screen prompts. For more information on **custom**, refer to the *SCO XENIX System V System Administrator's Guide*.

5. You are instructed again to insert the floppy disk. Because this volume is already in the drive, press ⟨Return⟩.

6. When you are finished installing the Toolkit, quit out of **custom** by entering **q** at the main menu.

Upgrading an SCO XENIX System V Toolkit

If you are upgrading from a previous release of the Toolkit used with the SCO XENIX System V operating system, follow this procedure:

1. Use **custom** to remove the old Toolkit.

2. Remove the */etc/perms/petkit* file.

3. Use **custom** to install the new Toolkit.

International Considerations

When installing the Toolkit in an international environment, you should edit the */usr/bin/docut* shell script to change the LANG variable at the start of the file to the language that your system is using.

Chapter 2

Cutting the Distribution

Introduction

Before you use the utilities described in this guide, it may be helpful to understand how they work. The utilities contained in the Toolkit are introduced here, along with short descriptions of their uses and relationships to each other.

This chapter outlines the entire installation procedure, describing how and when to use each utility in the Toolkit. Also presented here are suggestions for organizing your distribution files. The tasks explained here are meant as suggestions only; they are not necessary for the completion of the remaining procedures in this guide, which help you to prepare your distribution for **custom** installation.

This chapter then shows the procedures for cutting a **custom**(ADM)-installable distribution using **docut**(SCO).

The following topics are discussed as sections:

- the Toolkit software and organizing your distribution files
- setting up a distribution hierarchy
- setting distribution variables
- running the **docut** utility to create a permissions list and to copy files for use by **mkcuts**(SCO)
- running **docut**(SCO) more than once
- editing the permlist
- running **mkcuts**(SCO) to create a sums list or a disk image
- running **mkflops**(SCO) to put a disk image on floppy disk
- using other Toolkit utilities

Toolkit Software

The Toolkit consists of a set of utilities and files that prepare and move your distribution files from hard disk onto removable media. The Toolkit utilities help you to create a **custom**-installable application. This means that you can place your files on some form of storage media such as floppy disks or cartridge tapes, and then install those files onto another computer using the **custom** utility. This section briefly introduces these utilities and describes their relationships to each other as well as to **custom**.

The Toolkit consists of the following utilities:

Utility	Description
diskimage(SCO)	creates disk images for rapid copying of the distribution files.
docut(SCO)	creates an application distribution.
fdfit(SCO)	fits file archives onto media volumes.
hocheck(SCO)	compares permlist with current and past distributions.
mkcuts(SCO)	makes **custom**-installable (application distribution cutting tool).
mkflops(SCO)	creates disks from disk images.
mkperm(SCO)	makes a product permission list.
pkgsize(SCO)	updates size information for package in special permlist comment.
volno(SCO)	updates volume number information for files.

Each of these utilities is described in the manual pages in Appendix A.

Optional Utilities

The following utilities are optional and their use is not recommended for generating a distribution. These utilities are provided for use by the Toolkit programs themselves and are not necessary for development purposes:

diskimage(SCO)	**mkperm**(SCO)
fdfit(SCO)	**pkgsize**(SCO)
hocheck(SCO)	**volno**(SCO)
mkmaster(SCO)	

For all the product features described in Chapter 1 in the "Product Features" section, you only need to use **docut**(SCO), **mkcuts**(SCO), and **mkflops**(SCO). This chapter describes the use of these three utilities. Additional information concerning the use of the optional utilities is described at the end of this chapter in the "Using the Optional Utilities" section. Again, use of the optional utilities is complex, fraught with the possibility of errors, and not recommended.

Using the Utilities

The Toolkit utilities are used as shown in the figures that follow.

When creating a **custom**-installable distribution, start by running **docut**. If your distribution does not require that you generate a "sums list" to verify that changes have occurred between releases, or if you do not need to create "disk images" for rapid copying of your product, just using **docut** completes your distribution.

Alternatively, if a sums list or disk images is required, run **docut** followed by the other section (SCO) utilities.

All Toolkit utilities are run from the *root* login. Log in as *root* before starting any of the work described in this chapter.

Before starting to run any of the Toolkit utilities, create a directory in which you will copy the distribution files. Because all of the Toolkit utilities are run from the *root* login, the permissions modes should be set to at least 700 with *root* user and group ownerships. Once this directory is created, create a *source* directory with the same permissions and ownerships. In this guide, the directory is called *source*, but you can give it any name you wish.

Simple Distribution

Figure 2.1 shows how to create a simple distribution with a single package. This type of distribution lets you compress files, but it does not let you produce a sums list or a disk image.

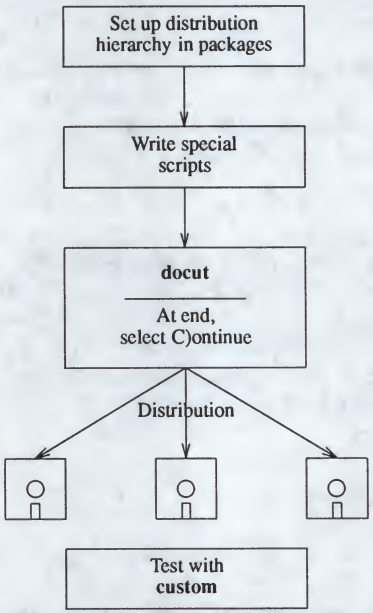


Figure 2.1: Simple Distribution

To prepare a simple distribution, follow these steps:

1. Set up a distribution hierarchy relative to your source directory. For detailed information on locating files in your source directory, refer to "Organizing Distribution Files" in this chapter.

2. Cut the distribution by running **docut**(SCO). Refer to "The docut Utility" in this chapter.

3. Test the distribution by running **custom**. For detailed information, refer to Chapter 3, "Installing an Application."

Simple Distribution with Packages

Figure 2.2 shows how to create a simple distribution with packages.

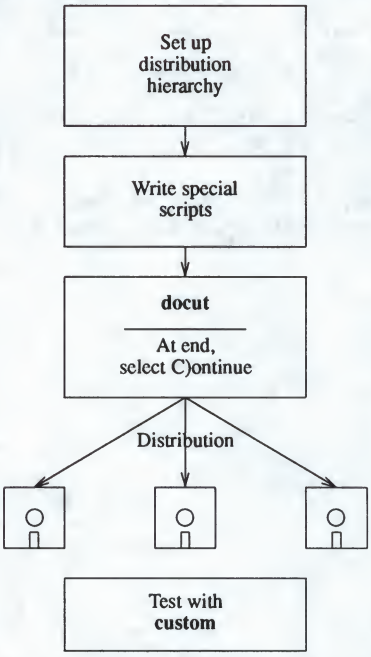


Figure 2.2: Simple Distribution with Packages

To install an application that contains more than one package, follow these steps:

1. Set up a distribution hierarchy. For detailed information, refer to "Organizing Distribution Files" later in this chapter.

2. Write special scripts (for initialization, preparation, and removal) as needed.

    - A typical initialization script for this type of installation could set the system name in a data file.

    - A typical removal script could remove any system file modifications required by the application if the user decides to remove your application (with **custom**).

- A typical preparation script could eliminate files from the permissions list that are not needed on the particular type of system on which the software is being installed.

For detailed information, refer to Chapter 4, "Writing Installation Scripts."

3. Cut the distribution files by running the **docut**(SCO) utility. For detailed information, refer to "The docut Utility" in this chapter.

4. Test the distribution by running **custom**. For detailed information, refer to Chapter 3, "Installing an Application."

Distribution with a Sums List

Figure 2.3 shows how to create a distribution when generating a sums list.

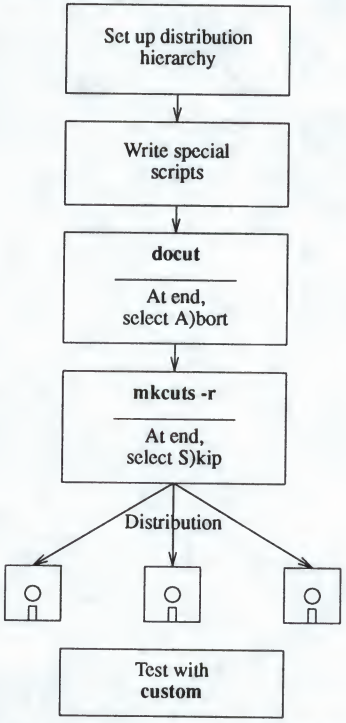


Figure 2.3: Distribution with Sums List

Follow the installation instructions for the previous section, "Simple Distribution with Packages."

Distribution with a Disk Image

Figure 2.4 shows how to create a distribution that produces a disk image.

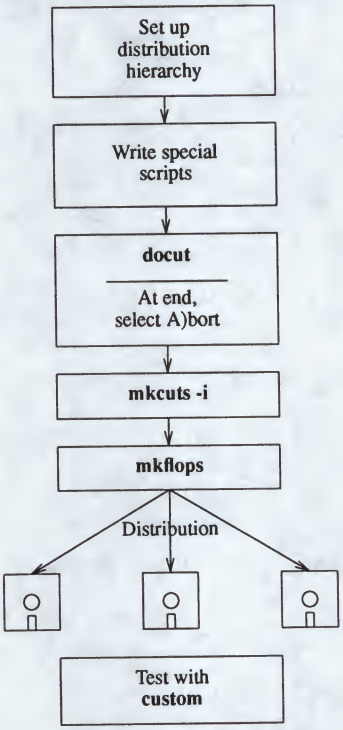


Figure 2.4: Distribution with Disk Image or Sums List

Follow the installation instructions for the previous section, "Simple Distribution with Packages."

Distribution Containing a Device Driver

Figure 2.5 shows how to create a distribution that contains a device driver.

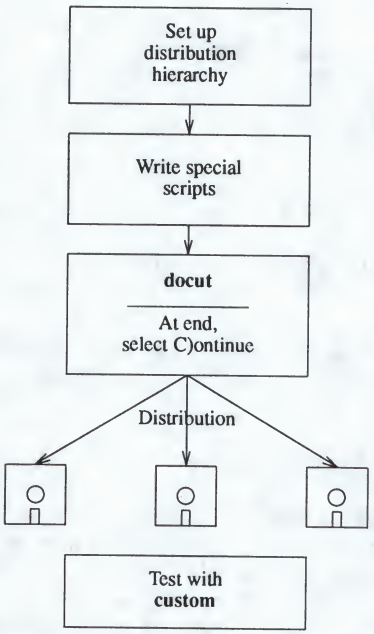


Figure 2.5: Simple Distribution with Packages

To prepare an installable device driver (IDD) installation, follow these steps:

1. Set up a distribution hierarchy. For detailed information, refer to "Organizing Distribution Files" later in this chapter.

2. Write special scripts (for initialization, preparation, and removal) as needed. A typical initialization script for a device driver performs the following tasks:

   - loads the link kit (if not present)
   - installs the new driver using the link kit (using the **configure** utility)
   - relinks the kernel with the new driver

   For detailed information, refer to Chapter 4, "Writing Installation Scripts," and Chapter 5, "Installing Device Drivers."

3. Cut the distribution by running **docut**(SCO). For detailed information, refer to "The docut Utility" in this chapter.

4. Test the distribution by running **custom**. For detailed information, refer to Chapter 3, "Installing an Application."

Software Overview

A short synopsis of the use of the product engineering utilities follows. In addition to the Santa Cruz Operation (SCO) section utilities are other utilities that are described in other reference manual sections depending on your operating system type. Refer to Chapter 1 for a list of the correct manual page sections for SCO XENIX System V. More information on the section (SCO) utilities is found in the next chapter and in the manual pages for the utilities in Appendix A. Refer to the reference manuals supplied with your operating system for more information on System Administration (ADM) section utilities.

The utilities are:

custom(ADM)
: installs a product distribution. Using the permissions list (called a "permlist"), **custom** copies the files from the distribution media, such as a floppy disk, to the system on which the product is being installed. A permlist is a file that describes where each file in the distribution should be located when the files are CUT (copied) to the computer on which the software product is being installed. The permlist also describes the permission modes that are assigned to each file as it is installed. The permlist is created by the **mkperm**(SCO) utility.

diskimage(SCO)
: reblocks the distribution files into a file that is the size of the output disk media so that the file be copied directly onto a floppy disk using the **mkflops** utility. This utility is called by **mkcuts**(SCO).

docut(SCO)
: creates a **custom**-installable distribution. Always run first to either prepare the distribution automatically or to generate the **mkmaster** script for manual control, this utility prompts you for information required to find and create a distribution. The **docut** utility executes the **mkperm** utility to create a permissions list.

fdfit(SCO)
: fits the distribution files onto the floppy disk volumes in the most efficient manner for each file in the permissions list. This utility ensures that a file is not split between floppy disks, unless the size of the file exceeds the size of a floppy disk. This utility is called by **docut**(SCO) and **mkcuts**(SCO).

fixperm(ADM)
: corrects file permissions and ownerships in a distribution. Using the permlist, **fixperm** resets permission modes back to the value designated in the permlist. This utility is called by **docut**(SCO) and **mkcuts**(SCO).

hocheck(SCO)
: compares a permissions list against the actual files in the distribution directories. Filenames containing discrepancies are noted and placed in a directory. The permlist is created with the **mkperm** utility and can be edited with a text editor such as **vi**(C), although care must be taken to ensure that changes to the file are not overwritten when other section (SCO) utilities are run. **hocheck** ensures that the permlist is correct.

mkcuts(SCO)
: uses **fixperm** to extract the information in the permlist to determine which files are on the distribution. It then takes this information and calls **fdfit, fixperm, pkgsize,** and **volno** to create the distribution. **mkcuts** has the capability to create a sums list and disk images. **mkcuts** creates a label file in the distribution hierarchy.

mkflops(SCO)
creates output floppy disks using disk image files generated by **mkcuts**(SCO) or **diskimage**(SCO).

mkmaster(SCO)
creates a standard distribution without the additional features that **mkcuts** provides. **mkmaster** creates a label file, uses the permissions list to verify the distribution hierarchy, compresses the distribution (if requested), executes **fdfit** to create a list of files for each media volume, and executes **volno** to update the permlist with the volume number information. This utility is called by **docut**(SCO).

mkperm(SCO)
generates a permissions list for all files in the distribution. The PERMISSIONS LIST includes all files in the distribution along with their file permissions and relative pathnames. Used during installation, the list controls which files from your distribution get placed onto media. This utility is called by **docut**(SCO).

pkgsize(SCO)
updates the total disk space required by each package listed in the permlist. A PACKAGE is a logical portion of the entire application (help files or font files, for instance). A package lets you selectively choose which portions of the application to install with **custom**. After running **pkgsize**, you can go back to the permissions list and find the correct byte size listed for each package name. This tells you how much disk space is used by each package of your distribution. This utility is called by **docut**(SCO) and **mkcuts**(SCO).

volno(SCO)
generates a media volume number for each distribution file listed in the permissions list. After running this utility, you can go back to the permissions list and find a volume number listed beside each file that tells you on which volumes your files are located. This utility is called by **docut**(SCO) and **mkcuts**(SCO).

Organizing Distribution Files

Because the files of a product may be installed (and removed) according to package organization, they should be grouped so that they follow a logical pattern that allows the user maximum flexibility in the installation. Likewise, packages should be grouped in logical patterns within each product.

Several conventions concern file locations:

- Store all front-end shell script files that are accessed directly by the user in */usr/bin*. A front-end shell script contains code that calls binary and other shell script files. The binary files that are called should be stored in */usr/lib/<product>*. Binary files should not be stored in */usr/bin*. Keep the number of the files placed in */usr/bin* to a minimum because, if this directory gets too full, it can slow the system down. If possible, the binary files should be relocatable. If you use an alternate location, indicate the new location with a shell environment variable.

- Store all supporting files in special directories located in */usr/lib*, and name each library directory for a specific application after that application. For example, supporting files for SCO Professional are located in the subdirectory */usr/lib/pro*. If possible, these files should also be relocatable.

- Store any sample files written for demonstration or for educational purposes, such as tutorials, in the application's library directory. Files such as these, and information files that are temporary, may be grouped together in a separate package that is optionally installable. This way, information is available to the user, but is removable if the user wants to conserve more disk space.

Setting Up a Distribution Hierarchy

Before you can cut your distribution, you need to answer this question:

Should I divide my product into packages?

A package is a group of files that can be installed and removed independently from the rest of your product. If your product naturally breaks down into separate components that can be used on their own, you may want to divide it into packages. This gives users the option of installing only those parts of your product that they need.

If your answer is "no," to the above question, put all of your source files directly in the *./source* directory.

If your answer is "yes," you must create some subdirectories in your *./source* directory. The names you give to the subdirectories are the names of your packages, and they must be all in upper-case letters.

Here is an example source hierarchy without packages:

```
./source/usr/bin/gizmo
./source/usr/bin/ultragiz
./source/usr/lib/gizmo/gizmo.lib
./source/usr/lib/gizmo/ultragiz.lib
```

The same source could be organized into packages like this:

```
./source/GIZ/usr/bin/gizmo
./source/GIZ/usr/lib/gizmo/gizmo.lib
./source/ULTRAGIZ/usr/bin/ultragiz
./source/ULTRAGIZ/usr/lib/gizmo/ultragiz.lib
```

Init, Prep, and Removal Scripts

Initialization (init), preparation (prep), and removal scripts must be placed in the correct locations in the distribution hierarchy. Refer to Chapter 4, "Writing Installation Scripts," for more information on the scripts described in this section. In addition, refer to the sample scripts supplied with the Toolkit software:

/usr/lib/petkit/samples/init.sample
/usr/lib/petkit/samples/prep.sample

Additional sample scripts are provided for installable device driver (idd) products.

Follow these guidelines to place the init, prep, and removal scripts in their correct locations:

- **Initialization script**

  Init scripts are run after package files are extracted. Multiple initialization scripts are permitted for a product. The init scripts should go in the *./source/tmp/perms* directory. If your product is divided into packages, locate the init scripts subordinate to the package directory. The name must be of the form *init.PKGNAME*. If your product is not divided into packages, the name should be of the form *init.productname*, where *productname* is the abbreviated product name (also known as the prd value, or PRODPRD).

- **Preparation script**

  Prep scripts are run before any product files are extracted. One preparation script is permitted for a product. The prep script should go in the *./source/tmp/perms* directory. If your product is divided into packages, locate the prep script subordinate to a package directory. The name must be of the form *prep.PKGNAME*. If your product is not divided into packages, the name should be of the form *prep.productname*, where *productname* is the abbreviated product name (also known as the prd value).

- **Removal script**

  One removal script is permitted, but not required, for a product. The removal script is placed in the *./source/usr/lib/custom* directory. The name must be of the form *productname.rmv*, where *productname* is the abbreviated product name (also known as the prd value).

Distribution Hierarchy Checklist

Run through this checklist to make sure that your distribution hierarchy is set up properly:

- ❒ Is your product divided into appropriate packages?
- ❒ Are the package names in all uppercase letters?
- ❒ Do your files have the correct path? Remember that the path of your files below *./source* corresponds to the absolute path at which they are installed. For example, both

  ./source/usr/bin/gizmo

  and

  ./source/GIZ/usr/bin/gizmo

  are installed as:

  /usr/bin/gizmo

- ❒ Do your files have the correct ownership and permissions? (If it is inconvenient to set these values by altering your source files, you can wait and edit the permission list later. See the section "Editing the Permlist" for details.)

Setting Distribution Variables

An additional aspect of setting up your distribution hierarchy is to copy in the files that contain variables that can be set to reflect the needs of your distribution.

The **mkcuts**(SCO) utility uses a set of variables to which you can assign values. To set the variables:

1. Copy the */usr/lib/petkit/site_variables* file to a new directory that is parallel to the distribution *./source* directory. Set the permissions modes for *root* ownerships and for *root* write access only.

2. Make the *site_variables* file executable.

3. Make the changes to *site_variables* that are described in this section. Refer to the *site_variables* file for the values of each variable. This file contains these variables, shown here in alphabetic order:

BLOCKING	PRODPRD
CDISTDIR	PRODREL
DEVICE	PRODSET
DISTDIR	PRODTYP
FORMAT	PRODUPD
MISCDIR	SERIALIZE
NOTAR	SERIALKEY
OTHER_MEDIA	VOLSIZE
PERMLIST	

4. Edit the */etc/default/petkit* file, if necessary, to ensure that IMAGEDIR and SUMDIR are correct for your needs. Refer to the file for the default values.

Using the comments in the file as a guide, set the following variables in the *site_variables* file (setting the */etc/default/petkit* variables is described thereafter).

/usr/lib/petkit/site_variables File Variables

1. Enter the full name of your distribution's product for the PRODSET variable. Use quotations marks (" ") around a name that has more than one word. The string can contain up to 99 characters. The permlist is modified to match this variable. Use the following syntax:

   PRODSET=<"*full product name*">

2. Enter a short, or abbreviated, product name for the PRODPRD variable. The permlist is modified to match this variable. Use the following syntax:

   PRODPRD=<*shortened product name*>

3. Enter the code of the target machine and environment type for the PRODTYP variable. Refer to "The docut Utility," step 6, for a list of the target system variables. The permlist is modified to match this variable. Use the following syntax:

   PRODTYP=<*target machine and environment type*>

4. Enter the current release number and iteration letter for the PRODREL variable. The permlist is modified to match this variable. Use the following syntax:

   PRODREL=<*current release number*>

5. Enter the name of the device that is archiving the distribution for the DEVICE variable. Refer to the */etc/default/tar* file for full listings of device names. Use the following syntax:

   DEVICE=<*archiving distribution's device name*>

6. Enter the byte size of each distribution volume for the VOLSIZE variable. Here are the most commonly used device names and sizes:

Device	Volume Size
/dev/rfd096ds15	1200K (1.2 Mbytes)
/dev/rfd096ds9	360K
/dev/rfd0135ds9	720K
/dev/rfd0135ds18	1440K (1.4 Mbytes)

Use the following syntax:

```
VOLSIZE=<distribution volume size>
```

The number used here depends on the size of one media volume on which the product is distributed. Suffixes after the number multiply the number accordingly: *k* multiplies the number by 1024; *b* by 512; *w* by 2.

7. Enter the blocking factor of the archiving device. This number represents the number of blocks that are read from or written to the device each time a read or write is requested from a user program. Refer to the */etc/default/tar* file for this value. Set the BLOCKING variable, if required, using this syntax:

```
BLOCKING=<blocking factor>
```

8. Enter the complete command needed to format a volume. Make sure to use quotation marks (" ") around the command. For the FORMAT variable, use the following syntax:

```
FORMAT=<"complete format command">
```

9. If in **mkcuts** you wish to hold back some files from being copied to the output media, set NOTAR to the names of the files. Use this syntax:

```
NOTAR=<"pathname pathname">
or
NOTAR=<pathname>
```

10. Enter the name of the directory where the distribution files are placed for use by **mkcuts**. The syntax is:

```
DISTDIR=<pathname>
```

11. Enter the name of the directory where temporary files are stored by the Toolkit utilities:

    ```
    MISCDIR=<pathname>
    ```

12. Enter the device name (basename only) if your device is not one of the following devices that are assumed by **mkcuts** and **mkflops**:

Device	Media Type
*/dev/*48ds9*	48dsdd
*/dev/*96ds8*	96dsdd
*/dev/*96ds15*	96dshd
*/dev/*135ds9*	135dsdd
*/dev/*135ds18*	135dshd
*/dev/*rct0*	tape
*/dev/*rctmini*	mini

    The syntax is:

    ```
    OTHER_MEDIA=<device-name>
    ```

13. Enter the update string, if applicable, for the PRODUPD variable. This string is two characters that must start with "U" (for update). Use the following syntax:

    ```
    PRODUPD=<update string>
    ```

    If you are not cutting an update, it is crucial that you leave this field blank.

14. These two variables are reserved for future use:

    ```
    SERIALKEY
    SERIALIZE
    ```

15. Enter the name of a directory where the permissions list is stored if other than the default path:

    ```
    PERMLIST=<pathname>
    ```

16. Enter the name of a directory where the compressed distribution files are to be stored. Refer to the "mkcuts -c (Compress) Option" section later in this chapter for more information. The syntax is:

    ```
    CDISTDIR=<pathname>
    ```

/etc/default/petkit Variables

1. Enter the name of the directory from which the disk image directory hierarchy is created. Refer to the "mkcuts -i (Disk Image) Option" section later in this chapter for more information. The syntax is:

   `IMAGEDIR=<`*pathname*`>`

2. Enter the name of the directory where the sums list is stored. Refer to "mkcuts -s (Sums List) Option" later in this chapter for more information. The syntax is:

   `SUMDIR=<`*pathname*`>`

If you want to make changes to any of the values that you set up, edit the appropriate parameter file.

The docut Utility

A timesaver for developers, the **docut** utility handles almost all of the tasks necessary to make a **custom**-installable application out of a group of files.

What docut Cannot Do

The **docut** used by itself has limitations. If you require any of the items listed below, use **docut** to start followed by **mkcuts**. The **docut** utility cannot generate:

- **a sums list for the files in your product**

  The sums list identifies which files have changed between cuts of your product. To create a sums list, use **docut** to generate the permlist and then run **mkcuts** to produce the sums list.

- **disk images of your distribution**

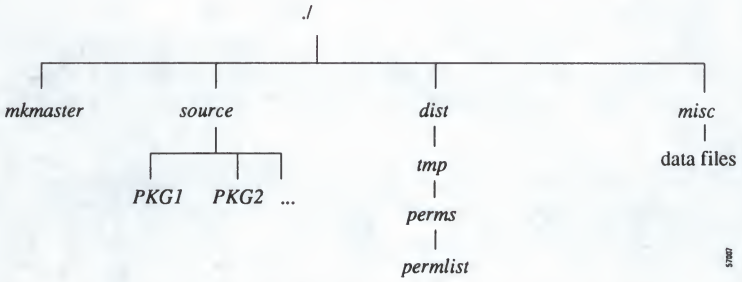
  The disk images quickly create extra copies of your product. To create disk images, use **docut** to generate the permlist, then use **mkcuts** to produce the disk image. Finally, use **mkflops** to copy the disk image onto your floppy disk distribution media.

Running the docut Utility

Once you have copied your distribution files and directories into the *source* directory, you are ready to cut the distribution with the **docut** utility.

The **docut** utility creates distribution volumes that are **custom** installable.

The **docut** utility uses the following directory hierarchy:



The **docut** utility creates the *mkmaster* file, the *dist* directory, and the *misc* directory. The *source* directory is one that you create before running **docut**. The *PKG* (package) directories are also ones that you create, but are optional.

Step 1: Before Running docut

Perform the following measures before using **docut**:

1. Create the directory that contains the *source* directory hierarchy and the *source* directory itself. These directories can be anywhere in the filesystem. Put the files that make up your product in *source* directory, creating subordinate directories as needed. If your distribution has packages, locate the package directories immediately below the *source* directory.

2. Copy the init and prep scripts to the *./source/tmp/perms* directory, and copy the removal script to the *./source/usr/lib/custom* directory. If your distribution has packages, put these scripts in a single package of your choice.

3. If you intend to use **docut** to cut your distribution media, ensure that you have floppies or other media on hand before you run **docut**. The floppies need not be formatted; you can format them while running **docut**.

4. Log in as *root*.

5. Make sure that every file has a path that corresponds to its destination. For example, if you have a file called *gizmo* that should be placed in */usr/bin* when your product is installed, put the file in *./source/usr/bin/gizmo*.

6. Make sure that the permissions and ownership are correct for every file.

7. Display the */etc/defaults/tar* file to determine the size of each distribution volume, its blocking factor, and the device name. Use any display command, such as **more**(C) or **cat**(C). For example, a few lines of a sample */etc/default/tar* file are as follows:

```
#     device                  block   size      tape
archive0=/dev/rfd048ds9       18      360       n
archive1=/dev/rfd148ds9       18      360       n
archive2=/dev/rfd096ds15      10      1200      n
archive3=/dev/rct0            20      150000    y
```

   From this listing, for a 1.2 Mbyte floppy disk (archive2), the size is 1200 bytes, the blocking factor is 10, and the device name is */dev/rfd096ds15*. Write this information down because you will need it in step 9.

8. **cd**(C) into the directory that contains *./source*.

Step 2: Run docut

If your application is not divided into packages, run the **docut** utility using the following syntax:

docut

If your application is divided into packages, use this syntax:

docut -p *./source*

where *./source* is the name of the directory you set up for your distribution files. (For more information, see the preceding section, "Setting Up a Distribution Hierarchy.")

If you want to start **docut** fresh so that none of the previous values are retained, use the **-e** flag:

docut -e

If you want to compress the distribution files using the **compress**(C) command, add the **-c** flag:

docut -c
or
docut -c -p *./source*

Note

Only versions of **tar**(C) that have the **C** option can uncompress these files. Most SCO XENIX System V and SCO UNIX System V systems do not support this option. If the target system cannot support this capability, do not use **docut -c**.

You are now prompted for information about your product. You may not see all of the prompts listed here (depending on how you set up your distribution hierarchy and which options you use with **docut**), but they are all provided for your reference. The information requested in these prompts can be provided to **docut** with a series of variables. These variables are explained in "Setting Distribution Variables" later in this chapter. To help identify the prompt with the variable, if a variable is associated with a prompt, then the name of the variable is shown in brackets [ ] after the explanation of the prompt.

Step 3: Enter a Product Name

The **docut** utility prompts are generated by the **mkperm**(SCO) utility and are as follows:

```
Please enter the product name (e.g. toolkit):
```

Enter an abbreviated name for your product. This name can be up to 14 characters in length, must adhere to file naming conventions, and must be a single word. [PRODPRD]

Step 4: Enter the Application Name

```
Please enter the application name (e.g. SCO 386 Toolkit):
```

Enter the complete description of the product. [PRODSET]

Step 5: Enter the Release Number

```
Please enter the release number or press <Return>
for the default value of:
```

Enter the release number and iteration level of the product or press <Return> to select the default. An example, would be "3.0.1a". [PRODREL]

Step 6: Enter the Target System Value

```
Please enter the target system value or press <Return>
for the default value of:
```

Enter the value for the target system or press <Return> to select the default. A list of target names is as follows: [PRODTYP]

Type	Description
286AT	IBM® PC AT® or compatible
286BM	Bull Micral 60™
286ESDI	286AT class with SMS OMTI 8620 or 8627 controller
286GT	All machines in the 286GT class
286HP	Hewlett-Packard Vectra™ 45945A
286MC	Micro Channel™
286OM	Iomega® Bernoulli Box®
286PS	IBM PS/2® Model 50 or Model 60
286XI	Apricot XEN-i™
386AT	Compaq DeskPro 386™ or compatible
386ESDI	386AT class with SMS OMTI 8620 or 8627 controller
386GT	Machines in 386AT, 386SCSI, and 386ESDI classes
386MC	Micro Channel
386PS	IBM PS/2 Model 80
386SCSI	386AT class with Adaptec™ SCSI host adapter
6300+	AT&T™ 6300+
86XT	IBM PC XT® or compatible
dosn86	Software to run on DOS
k286	For n286 machines, specific to the SCO XENIX System V kernel
k386	For n386 machines, specific to the SCO XENIX System V kernel
ku386	For u386 machines, specific to the SCO UNIX System V kernel
n286	For 286 or 386 machines running SCO XENIX System V
n286.5	Like n286, except must run SCO XENIX System V
n386	For 386 machines running SCO XENIX System V or SCO UNIX System V
n86	For any SCO XENIX System V or SCO UNIX System V machine
other	Other environment code not on this list
sun3	SunOS™ 3.5, sun3 68020/68030 CPU
sun386i	SunOS 4.0, sun386i 386 CPU
sun4	SunOS 4.0, sun4 SPARC™ CPU, SPARCstation™1
u386	For machines running SCO UNIX System V

Step 7: Describe the Source Directory Hierarchy

The following sections delineated by the ❖ symbol contain two different pathways depending on whether or not you run **docut** with the **-p** option to indicate that your distribution contains packages. Each pathway is described completely. Most of the prompts are the same; only the context differs.

❖ Distribution Without Packages

These prompts are shown when **docut** is run without the **-p** option. In the context of these prompts, a *package* is considered to be the full distribution. Eventually within these prompts, you can augment this vision of your distribution to specify additional package names. Then *package* comes to mean whatever part of your distribution that you are currently describing.

```
Please enter the directory where the package lives:
```

Enter the relative pathname for the source directory. Typically, this is *./source*. In this context, *package* refers to your full distribution. Then the next prompt is displayed:

```
Do you wish to enter another package directory? (y/n)
```

Enter **y** to enter the names of other packages in your product. The previous prompt is displayed and then this one until you have entered all the packages in your distribution. These two prompts permit you to either specify the location of a single package distribution or to enter all the packages as though the **-p** option were specified.

```
Please enter the package name associated
with the directory <source>:
```

Enter the package name, in all caps, for the distribution.

```
Please enter a short description for the
package <package name>:
```

Enter the full name of the product or a full description of the package.

Skip now to step 8 to complete the prompts.

❖ Distribution With Packages

This is the first prompt shown when **docut -p** is specified:

```
Please enter a short description for the
package <package name>:
```

Enter the full name of the product or a full description for the package.

Note

This is where the two option pathways diverge. The last steps appear the same as shown regardless of whether or not the **-p** option is specified.

Step 8: Enter the Output Filename

You see the following message:

```
Please enter the name of an output file or press <Return>
for the default of ./dist/tmp/perms/<permlist>:
```

Press <Return>. Always pick the default for this prompt.
The following messages are then displayed:

```
Creating distribution hierarchy.

Creating mkmaster script.

Please enter the device name for archiving
the distribution or press <Return>
for the default value of /dev/rfd096ds15
or enter q to quit:
```

Enter the device name, or press <Return> to select the default.

Step 9: Enter the /etc/defaults/tar Information

The next few prompts rely on the information that you obtained in step 1 from the */etc/defaults/tar* file.

```
Please enter the size of each distribution volume
or press <Return> for the default value of 1200K
or enter q to quit:
```

Enter the volume size, or press <Return> to select the default.
[VOLSIZE]

```
Please enter the blocking factor if required, 0 if not,
or press <Return> for the default value of 10
or enter q to quit:
```

Enter the blocking factor, if different from 10, or press <Return>.
[BLOCKING]

```
Please enter the complete command to format a volume
or press <Return> for the default value
of ''format /dev/rfd096ds15''
or enter q to quit:
```

Use the device name that you found in step 1 for the distribution media device. Refer to the **format**(C) manual page for information about the options to the **format** command. Press <Return> to select the default, or type in the complete format command of your choice. [FORMAT]

Step 10: Make Media Choices

The following informative messages are displayed:

```
Executing mkmaster.

Setting permissions...

Updating package sizes ...

Creating contents listings ...

Updating volume numbers...

Creating number master...
```

Then this prompt is displayed:

```
C)ontinue, A)bort or F)ormat:
```

Enter **c** to complete the distribution with the files being copied to the distribution media using **tar**(C).

Enter **a** to not copy the files to the distribution media ar this time. In this context, "Abort" is misleading. All of the byproducts of **docut** are completed, such as the permlist and the copying of the files to the *./dist* directory. "Abort" means only that you do not wish to copy the files to the output media at this time. Select Abort if you are creating a disk image or a sums list. Then call **mkcuts** to complete the distribution. Refer to "The mkcuts Utility" for more information. Abort also abandons the current volume and lets you skip to the next one. Abort makes it easy to skip to a specific volume if you need to recut a single volume (this is permissible as long as the permlist has not changed and file sizes did not been change).

Enter **f** to format a floppy disk volume before archiving any files.

Choose the appropriate option, and press <Return>. When all volumes are handled, **docut** exits.

Running docut More Than Once

The **docut** utility keeps data files that store the product specific information that you enter the first time they are run. This information can be changed by re-running the program.

1. Enter the appropriate command again, this time with no flags:

   docut

2. You see the following list of data:

```
The following values have been read from the perms
data file:

Product name:
Full set name: " "
Release value:
Target machine:
Serialized files: " "

The following package information has been read:

Directory    Name  Description
---------    ----  -----------
sourcePKG1   This is package 1
sourcePKG2   This is package 2
sourcePKG3   This is package 3

Do you wish to use these values? (y/n)
```

If the values are correct, type **y** and press <Return>.

If you wish to change any of the values, type **n** and press <Return>. You are prompted for specific values to change, beginning with the release number.

Editing the Permlist

If it is inconvenient to set the ownerships and permissions in the source hierarchy, you can interrupt **docut** and edit the permlist yourself. Follow these steps:

1. Set up your distribution hierarchy and invoke **docut** as described in this chapter.

2. Towards the end of the **docut** procedure, this prompt is displayed:

```
C)ontinue, A)bort or F)ormat:
```

   Enter **a** to exit **docut**.

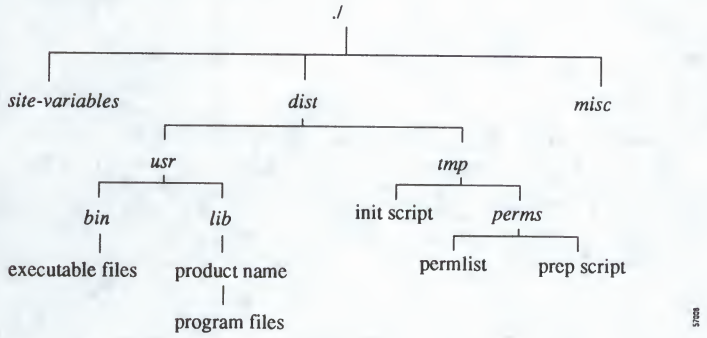
3. Edit the permlist (*./dist/tmp/perms/<PRODPRD>*) to reflect the correct ownership and permission values.

4. Run **docut** again to cut your master.

The mkcuts Utility

The **mkcuts** utility creates distribution volumes that are **custom**-installable.

The **mkcuts** utility uses the following hierarchy:



Before Running mkcuts

Before cutting a distribution using the **mkcuts** utility, follow these steps:

1. Ensure that you have floppy disks or other form of media on hand before running the **mkcuts** utility, because you will be prompted for them.

2. Set up a *source* distribution directory hierarchy and copied all distribution files into the subordinate directories. This directory also contains any init, prep, or removal scripts in the appropriate directories as was previously described earlier in this chapter.

3. Run **docut**(SCO) to create a permlist. **docut** also copies all files into a */dist* directory that is used by **mkcuts**.

4. Copy the */usr/lib/petkit/site_variables* to the directory containing the *source* and *dist* directories and give values to the variables in this parameter file.

5. Login as *root* and ensure that you have access to your distribution files and permissions list.

6. Make sure that you are in the directory containing the

Running mkcuts

Run the **mkcuts** utility, using the following syntax:

mkcuts *[option]*

The options are:

mkcuts

Option	Purpose
-c	creates compressed distribution files. This option requires that the CDISTDIR variable be defined in the *site_variables* file in the directory containing the *dist* directory.
-i	creates a disk image.
-s	creates a sums list.

The sections that follow show the messages and prompts that occur when you run **mkcuts**. The sections are listed as they appear in the program.

As long as one option is requested, **mkcuts** executes correctly. However, you can set implicit options for other tasks without specifying the command-line options through the use of variables that you set before running **mkcuts**. These variables, when set, trigger a prompt to which you enter "y" for yes, or "n" for no. The variables are:

Variable	Prompt
CDISTDIR	Do you wish compression?
IMAGEDIR	Do you wish to make images instead of floppies?
SUMDIR	Do you wish to make a sums list?

In each case, the directory pointed to by the variable must already exist.

Starting Messages

The starting messages for **mkcuts** are:

```
Creating PRODSET PRODUPD Release PRODREL master.

Setting permissions...

Updating package sizes...
```

mkcuts -c (Compress) Option

The **-c** option requires access to the CDISTDIR variable, which contains the directory in which the compressed files are stored.

Note

Only versions of **tar**(C) that have the **C** option can uncompress these files. Most SCO XENIX System V and SCO UNIX System V systems do not support this option. If the target system does not support this capability, do not use **mkcuts -c**.

If the CDISTDIR directory exists, then you are prompted to determine if you want to place your files in the specified directory:

```
Do you wish to re-compress the distribution hierarchy? (y/n)
```

Responding with "y" causes the hierarchy to be compressed and new label and permlist files to be added. This message is displayed:

```
percent% compression for:  file
```

Then this message is displayed:

```
Setting permissions in compressed hierarchy...
```

If you respond with "n" to the previous compression prompt, **mkcuts** does not compress the directory.

More Messages

The next set of messages is displayed:

```
Creating contents listings...

Creating MISCDIR...

Updating volume numbers...
```

mkcuts -i (Disk Image) Option

The **mkcuts** utility can create a disk image that can then be copied to floppy disk using the **mkflops** utility. This option requires that the IMAGEDIR variable contain the name of an existing directory from which the image file directories are built. The actual directory in which the image file is stored is four levels lower than whatever name you specify. The directory names are derived from the following variables. Even though this seems complex, the **mkflops** utility accesses this path directly if **mkflops** is called without arguments.

<IMAGEDIR>/<PRODPRD>/<PRODTYP>/<media-type>/<PRODREL>

The <*media-type*> directory tests the value of the DEVICE variable and then translates it into the following values:

DEVICE Value	**Resulting <*media-type*>**
*/dev/*48ds9*	48dsdd
*/dev/*96ds8*	96dsdd
*/dev/*96ds15*	96dshd
*/dev/*135ds9*	135dsdd
*/dev/*135ds18*	135dshd
*/dev/*rct0*	tape
*/dev/*rctmini*	mini

If a device cannot be found in the **mkcuts** translation table, "unknown" is entered. If "unknown" is undesirable, change the OTHER_MEDIA variable to a device name (basename). This name is used in the image directory name.

In addition to the image file, an additional file, *mapping*, is inserted in the same directory. This file contains a single line, which is a formatted date string.

If you are requesting a sums list in addition to the disk image, the sums information is also stored in the *mapping* file. Refer to "mkcuts -s (Sums List) Option" later in this chapter for more information on a sums list.

When the -i option is selected, the following message is displayed to provide the image filename and directory path:

```
Creating disk image in <directory/file>
```

Final mkcuts Prompt

If you did not specify the **-i** option, the following prompt is displayed:

```
C)ontinue, F)ormat, A)bort or S)kip to summing:
```

Enter **c**" **to tar** the distribution files to your output media. You are prompted successively for each volume master in your distribution.

Enter **a** to stop processing for the current master. Abort makes it easy to skip to a specific volume if you need to recut a single volume (this is only viable as long as the permissions list and file sizes did not change).

Enter **f** to format a volume before archiving files. The string entered in the FORMAT variable is executed to format your media.

Enter **s** to not **tar** your distribution files to your output media. All other processing is performed, except for this aspect. Skip completes the current volume and lets you skip to the prompt for the next master in your distribution.

Choose an option, and press <Return>. Continue to insert volumes as you are prompted for them.

You see this message:

```
Creating volume master...
```

When all volumes are handled, **mkcuts** exits.

mkcuts -s (Sums List) Option

This option creates a **sum**(C) **-r** checksum for each file in the distribution. This option depends on the SUMDIR variable being set to the name of the directory in which you want the output of the summing stored. The SUMDIR variable is described in the */etc/default/petkit* file.

The mkflops Utility

This utility copies disk images to floppy disks. Although **mkflops** has many options, the easiest way to run this utility is without options. Refer to the **mkflops**(SCO) manual page for more information about the options. Without any arguments, **mkflops** assumes that the image file or files are stored in the directory created by **mkcuts**, which has a pathname in this format:

<IMAGEDIR>/<PRODPRD>/<PRODTYP>/<media-type>/<PRODREL>

The **mkflops** utility can also generate a listing of all the information associated with your distribution. This listing is referred to on the **mkflops** manual page as the "Standard Release Form" and is activated with the **mkflops -r** or **-R** options.

Note

If you set the FORMAT variable in other utilities, note that this variable is not used in **mkflops**. The formatting options to **mkflops** prompt you to enter a floppy disk.

Using the Optional Utilities

The optional utilities in the Toolkit provide methods of cutting a distribution that are complicated and not recommended. These utilities reflect a methodology that led to the development of **docut**(SCO). The **docut** and **mkcuts**(SCO) utilities incorporate and supersede the optional utilities.

Refer to the manual pages in Appendix A for more information about how to run each utility. This section describes the context in which each utility is run.

The optional utilities are:

diskimage(SCO)	**mkperm**(SCO)
fdfit(SCO)	**pkgsize**(SCO)
hocheck(SCO)	**volno**(SCO)
mkmaster(SCO)	

Note

Before starting to use the optional utilities, familiarize yourself with shell script programming concepts. Study the source code for */usr/bin/docut* and */usr/bin/mkcuts* to see how the Toolkit utilities are used (except **hocheck**).

The optional utilities are run as follows:

diskimage(SCO)
: reblocks the files designated for a distribution master into a single file that is the same size as the output floppy disk. When the **diskimage** output file is copied to the output media, the original file format is restored with all original file permissions modes. The output from **diskimage** is analogous to a **cpio**(C) file in that many files are formed into a single file that, when restored, puts the files back into their original form. This utility is called by **mkcuts**.

Prior to running **diskimage**, perform the following steps:

1. Copy your files to create a *./source* distribution hierarchy.
2. Create a permlist with **docut** or **mkperm**.
3. Copy the files in the *./source* directory to the *./dist* directory.
4. Prepare the permlist for the distribution using **fixperm**.
5. Arrange the distribution files to fit onto the output media with **fdfit**.
6. Update the volume number with **volno**.

fdfit(SCO)

fits the files in a distribution onto the size of the distribution media to minimize the possibility of a file being spread between media volumes. This utility differs from **diskimage** in that instead of creating one large file as does **diskimage**, **fdfit** retains the original file images and just rearranges them. This utility is called by both **docut** and **mkcuts**. Prior to running **fdfit**, perform the following:

1. Copy your files to create a *./source* distribution hierarchy.
2. Create a permlist with **docut** or **mkperm**.
3. Copy the files in the *./source* directory to the *./dist* directory.
4. Prepare the permlist for the distribution using **fixperm**.

hocheck(SCO)

goes through the existing permlist and compares each entry with the actual files in the distribution. Compiling the information, **hocheck** then lists everything that it finds in files in a separate directory for your perusal. The utility lists all the devices, directories, files, links, and packages in the distribution but not in the permlist; also listed are the directories and files found in the permlist but not in the distribution. Empty files created by the permlist show up as missing, but this error can be ignored. Using this generated list, you can go back and edit the permlist or distribution, if necessary.

Instead of using **docut**, you can run **mkperm** to create a permlist and then run **mkcuts** to produce your master. When you are done running **mkperm**, you will need to copy your files from the *./source* directory where they were originally

organized, to the *./dist* directory for use by **mkcuts**. (These steps are done automatically by **docut**.) Because the copying of files is somewhat complicated and prone to possible problems, the **hocheck**(SCO) utility is provided to ensure that the files described in the permlist match those in the distribution directory. Again, this step is unnecessary if you use **docut**, and therefore, is not called by any other utility.

Prior to running **hocheck**, perform the following:

1. Copy your files to create a *./source* distribution hierarchy.
2. Create a permlist with **docut** or **mkperm**.
3. Copy the files in the *./source* directory to the *./dist* directory.

mkmaster(SCO)
reruns the information previously entered by using **docut**. Note, however, that when **docut** is run more than once, it does not reprompt you for the previously entered information unless you want to change a value. Therefore, executing **mkmaster** only saves you from having to press <Return> twice when using **docut**. Because this advantage is so minimal and requires remembering to add another step to your procedure, use of **mkmaster** is not recommended. **mkmaster** is called automatically from **docut**. Prior to calling **mkmaster**, copy the distribution files to the *./source* directory.

mkperm(SCO)
requests information to create a permissions list. This utility is called by **docut**(SCO). Prior to calling **mkperm**, copy the distribution files to the *./source* directory. All of the prompts requested when **docut** is executed are provided by **mkperm**.

pkgsize(SCO)
arranges files so that they can be copied onto the distribution media without files being spread between media volumes. Prior to calling **pkgsize**, perform the following:

1. Copy your files to create a *./source* distribution hierarchy.
2. Create a permlist with **docut** or **mkperm**.
3. Copy the files in the *./source* directory to the *./dist* directory.
4. Prepare the permlist for the distribution using **fixperm**.

volno(SCO)
updates the volume number in the permlist. This utility is called by **docut**(SCO) and **mkcuts**(SCO). Prior to calling **volno**, perform the following:

1. Copy your files to create a *./source* distribution hierarchy.
2. Create a permlist with **docut** or **mkperm**.
3. Copy the files in the *./source* directory to the *./dist* directory.
4. Prepare the permlist for the distribution using **fixperm**.
5. Rearrange the files to fit the size of the distribution media with **fdfit**.

Chapter 3

Installing an Application

Introduction

Now that you have prepared your distribution for installation by running the Toolkit utilities described earlier in this guide, you are ready to test the installation procedure with the **custom** utility. This chapter describes both the installation and removal procedures using **custom**. The following table lists the manual page sections for **custom** depending on the operating system in use:

Utility	SCO XENIX System V	SCO UNIX System V
configure	**configure**(C)	**configure**(ADM)
custom	**custom**(C)	**custom**(ADM)
fixperm	**fixperm**(M)	**fixperm**(ADM)

Refer to the reference manuals supplied with your operating system for more information on these utilities. Refer to Chapter 1 for listings of typical screens that are displayed when **custom** is executed.

Using information generated from the utilities described earlier in this guide, the **custom** utility installs the media volumes containing your distribution files. The utility checks to see that only the correct type of media is inserted, and also that the volumes are inserted in the proper order. Once **custom** has extracted the files, it runs the initialization script (if there is one). The init script may prompt the user for information. Then **custom** calls the **fixperm** utility to check the ownerships and permissions on the distribution files. Finally, the */tmp* files are removed, the action is logged in */usr/lib/custom/history*, and installation is complete.

The **custom** utility uses information generated by the **mkperm**(SCO) utility to do the following:

- Create the point-and-pick lists displayed during an installation.
- Ensure that the media used matches the product being installed.
- Check that sufficient space is available on the hard disk before installing the package.

The **fixperm** utility is invoked by **custom** to perform the following tasks:

- Provide information about the installation status of files and packages in the product (whether or not they are installed).
- Generate lists of files to be extracted by the **tar**(C) utility.
- Check that the correct files are installed, which includes verifying file types, ownership, and permissions.
- Create directories, zero-length files, devices, and file links.
- Prompt you for only those volumes that contain files you have requested.

Installing an Application

All of the procedures described earlier in this guide led you to this point: making your application **custom** installable. This section describes the procedure for installing an application, or packages from an application, with the **custom** utility. With your distribution volumes in hand, and a machine that contains the SCO XENIX System V or SCO UNIX System V **custom** utility, you are ready to test your installation.

There are differences between SCO XENIX System V and SCO UNIX System V versions of **custom**, both of which are described in the following sections.

Installing with SCO UNIX System V

Installation is done through the **custom** utility. Follow the steps outlined here to install an application:

1. Log in to the system as *root*.

2. Type **custom** and press ⟨Return⟩.

   To select a particular drive, type the following command:

   custom -m */dev/name*

   The device names are listed in the */etc/default/tar* file.

3. The initial **custom** menu appears. From the list of choices, select the appropriate option; for an application, select Install, and press ⟨Return⟩.

4. You see the following message:

   ```
   Select a Product
   ```

   Choose A New Product, and press ⟨Return⟩.

5. The Packages menu appears. Select Entire to install all of the packages. You see the following two messages:

```
Installing Custom Data Files...
creating file lists...
```

6. You are prompted to insert distribution volume 1. Insert volume 1 of your distribution into the correct device driver, and press ⟨Return⟩.

7. If you selected Packages, a display of package names appears. To install all the packages at once, select ALL, and press ⟨Return⟩. If you wish to install more than one package, select the packages using the <Space> bar, and press ⟨Return⟩.

8. You see the following message:

```
Extracting files...
```

   It takes several minutes for the files to be extracted from the volume.

9. You are prompted to insert distribution volume 2, if your distribution has more than one volume. Insert the next volume, and press ⟨Return⟩.

   Continue to insert volumes when prompted until all of the distribution files are extracted.

10. When all the selected packages of the distribution are extracted, **custom** runs the initialization script, if one is present in the permissions list. Then **custom** executes **fixperm**, which checks the file permissions, verifies that the files are installed in their correct locations, and creates directories and devices as specified by the permissions list. When this is done, the installation is complete.

    If you are finished with the installation, press any key to return to the **custom** menu. Select Quit and press ⟨Return⟩. Select Yes and press ⟨Return⟩ to exit.

Installing with SCO XENIX System V

Installation is done through the **custom** utility. Follow the steps outlined here to install your distribution:

1. Type **custom**, and press ⟨Return⟩.

   To select a particular drive, type the following command:

   custom -m */dev/name*

   The device names are listed in the */etc/default/tar* file.

2. The initial **custom** menu appears. From the list of choices, select the appropriate option; for an application, select "Add a supported product" and press ⟨Return⟩.

3. You are prompted to insert distribution volume 1. Insert volume 1 of your distribution into the correct device or drive, and press ⟨Return⟩.

4. The Action menu appears. From the list of choices, select the appropriate option; for an application, select "Install one or more packages," and press ⟨Return⟩.

5. You see the following message:

   ```
   Enter the packages to install
   ```

   The **custom** utility picks up the package names from the permissions list, which was generated by the **mkperm**(SCO) utility and is contained on volume 1 of your distribution.

   From the list of choices, select the appropriate option; to install all the packages at once, select ALL, and press ⟨Return⟩. If you wish to install more than one package, enter all the package names on one line, separated by spaces, and then press ⟨Return⟩.

   You are prompted again to insert distribution volume 1. At this point, simply press ⟨Return⟩.

6. You see the following message:

```
Extracting files...
```

   It takes several minutes for the files to be extracted from the volume.

7. You are prompted to insert distribution volume 2, if your distribution has more than one volume. Insert the next volume, and press ⟨Return⟩. Continue to insert volumes when prompted until you have installed all volumes.

8. When all the selected packages of the distribution are extracted, **custom** runs the initialization script, if one is present in the permissions list. Then **custom** checks the file permissions, verifies that the files are installed in their correct locations, and creates directories and devices as specified by the permissions list. When this is done, the installation is complete.

   The Action menu is again displayed to give you the option of performing other actions. If you are finished with the installation, enter **q** to quit the **custom** utility.

Removing an Application

After testing your distribution by installing it, you may now want to remove it from the system. This section describes the procedure for removing an application or packages from an application with the **custom** utility. There are differences between SCO XENIX System V and SCO UNIX System V versions of **custom**, both of which are described in the following sections.

Removing an Application from SCO UNIX System V

Follow these steps to remove an application:

1. Log in to the system as *root*.

2. Type **custom** at the prompt, and press ⟨Return⟩.

3. Select Remove, and press ⟨Return⟩.

4. The Packages menu appears, along with a list of each package in the set. Select the appropriate package, and press ⟨Return⟩. To remove all the packages at once, select ALL, and press ⟨Return⟩.

5. At this point, **custom** runs any appropriate removal scripts, and removes the files listed in the permissions list. After a few messages asking you to verify the removal and telling you that the files are being removed, you return to the **custom** menu.

   Select Quit, and press ⟨Return⟩. Select Yes, and press ⟨Return⟩ to exit.

Removing an Application from SCO XENIX System V

Follow these steps to remove an application:

1. Type **custom** at the prompt, and press ⟨Return⟩.

2. The initial **custom** menu appears, along with the following message:

   ```
   Select a set to customize
   ```

   Select the entry for your application, and press ⟨Return⟩.

3. The Action menu appears. Select "Remove one or more packages."

4. The Packages menu appears, along with a list of each package in the set. Select the appropriate package, and press ⟨Return⟩. To remove all the packages at once, select ALL, and press ⟨Return⟩.

5. At this point, **custom** runs any appropriate removal scripts, and removes the files listed in the permissions list.

   You return to the **custom** menu. Type **q** to quit.

Chapter 4

Writing Installation Scripts

Introduction

Three special-purpose scripts may be run during the installation procedure. The PREPARATION SCRIPT (prep) is run before any other installation activity; the INITIALIZATION SCRIPT (init) is run immediately after extracting the files from the media; and the REMOVAL SCRIPT (rmv) is run when a product is removed using the **custom**(ADM) utility.

This chapter describes the procedures for writing installation scripts. Also presented here are samples of scripts and conventions for writing them. The information contained in this chapter is not necessary for all installations. Refer to Chapter 2 for more information about how to use the Toolkit utilities.

Programming Conventions

Several conventions are established for SCO UNIX System V and SCO XENIX System V for the sake of consistency. These conventions are adopted by SCO and are presented here as suggestions to developers.

Consistent User Interface

The USER INTERFACE should be similar to that of the SCO XENIX System V or SCO UNIX System V and SCO applications to provide users with a consistent appearance and familiar interactive features. Several standard functions can be used to accomplish this. For example, the **getyn** function (refer to the sample scripts in Chapter 5 for a sample of **getyn**) prompts the user for a yes or no answer.

Note

The Toolkit distribution contains a series of sample scripts that you can use to study shell functions. These scripts are in both the */usr/bin* and */usr/lib/petkit/samples* directories. In addition, shell scripts are provided in the next chapter. More shell scripts containing example shell functions to use are in */usr/lib/mkdev/fd* and */usr/lib/mkdev/tape*, both of which are in the operating system.

Bourne Shell Programming Guidelines

The following are some Bourne shell programming guidelines to aid in your development:

- Use variables to make your script more legible (for example, exit values).
- Define variables explicitly at the start of the program (for example, PATH).
- Be aware of output of programs invoked; direct errors to *stderr*, and direct extraneous output to */dev/null*.
- Use a colon (:) as the first character of the first line to ensure execution as a Bourne shell script (as opposed to a *csh* script).
- Always specify an explicit exit value (0=success; 1=fail).

Writing an Initialization Script

Create an initialization (init) script for any special installation requirements, such as relinking the kernel or prompting the user for system-specific information. The init script is run during the installation process, right after the files are extracted from the media.

Here are a few typical procedures to include in an init script:

- Print the copyright message.
- Customize files and configuration after installation has taken place.
- Perform file linking, if necessary.
- Initialize files.

Here are the steps to take to create an init script:

1. Copy the init script into the *./tmp* directory. When the system is booted, all files in *./tmp* are removed.
2. Name the init script using this syntax:

   init.<productname>

   where *productname* matches the product variable listed in the permlist. (*productname* is also called the **prd** value, which is a variable listed in the permlist, and is the same value as PRODPRD in */usr/lib/site_variables*.) Or, if there are init scripts unique to each package, use this syntax:

   init.<pkgname>

   where *pkgname* matches the package name.
3. While a shell script is not the required format for the init script, it is recommended because shell scripts are powerful, portable, and easy to maintain.
4. Make sure that a zero (0) exit status is returned for successful initialization and a one (1) is returned in cases of failure, because the **custom**(ADM) utility checks the exit status of the initialization script.

5. When setting permissions for the init script, make sure that it is executable by **custom**(ADM) on the distribution as well as in the permlist. Set the permissions with this syntax:

   chmod 755 *init.<pkgname>*

6. This final step is necessary only if you run either **mkcuts**(SCO) or **mkmaster**(SCO) as a standalone utility. If you plan to run **docut**(SCO) or **mkperm**(SCO), then you can skip this step.

   Add a line to the permlist for the init script file, listing the init script in the same package as the files to which it is associated. For example, if your product has three packages, only one of which has an init script, then list the init script in only that package in the permlist. In this case, if the user chooses to install a single package, only the initialization script associated with that one package is invoked.

   Conversely, if you have a separate init script for each of your three packages, then these scripts must still be distributed in *./tmp,* each must have a unique name in the form *init.<pkgname>,* and each must be listed in the package with which it is associated.

   Finally, if you want the same init script to be run with every package in your distribution, it must be listed in every package in the permlist. In this case, if you use the **mkcuts**(SCO) utility, the init script appears on the floppies more than once.

Writing a Preparation Script

Create a preparation (prep) script for an activity that must take place before files are read from disk, such as saving the users' files before updating a product. The prep script is run only once: when the product is first installed, before any of the files (other than the permlist) are extracted from the media.

Prep scripts can be placed in any package subdirectory, or in the *./source* directory if the product is not divided into packages.

Here are a few typical procedures to include in a prep script:

- Check for pre-existing files to avoid overwriting.
- Check the version of the operating system or the CPU to determine if the target is acceptable.
- Ask the user miscellaneous questions before installation.
- Ask the users if they really want to overwrite the older version of the application that already sits on the hard disk.

Here are the steps to take to create a prep script:

1. Place the prep script in the *./tmp/perms* directory.
2. Name the prep script using this syntax:

   prep.<productname>

   where *productname* matches the product code name (the **prd** variable listed in the permlist, or PRODPRD in */usr/lib/site_variables*).
3. While a shell script is not the required format for the prep script, it is recommended because shell scripts are powerful, portable, and easy to maintain.

4. When setting permissions for the prep script, make sure that it is executable by **custom**(ADM) on the distribution as well as in the permlist.

5. Make sure that a zero (0) exit status is returned for successful initialization and a one (1) is returned in cases of failure, because the **custom**(ADM) utility checks the exit status of the prep script.

6. This final step is necessary only if you run **mkcuts**(SCO) as a standalone utility. If you plan to run **docut**(SCO) or **mkperm**(SCO), then you can skip this step.

   Add a line to the permlist for the prep script file. Make sure that it gets listed in the perm package. (Refer to step 6 in the section "Writing an Initialization Script.")

Writing a Removal Script

Create a removal (rmv) script for one of the following two cases: when some of the files associated with the application are not listed in the permlist (for example, they are created during installation) and, therefore, cannot be removed automatically by **custom**(ADM); or, when some special activity done in the init script, or some other setup script, needs to be "undone," such as adding information to a system file or changing a system parameter. In either case, the removal script is similar to the init script, except that it is run when the **custom**(ADM) utility's "Remove one or more packages" option is selected.

Refer to the sample removal script in "A Sample UNIX Removal Script" in the next chapter.

Here are a few typical procedures to include in a removal script:

- Remove */etc/rc* lines added during installation of the product (for example, remove file recovery code from */etc/rc*). Under SCO UNIX System V, installation scripts are in the */etc/rc.d* directory.

- Remove link line object files from *link_xenix* and invoke a recompile of the SCO XENIX System V operating system. (Under the SCO UNIX System V operating system, *link_unix* is not edited and is a list of object files is not required. The object files are brought together when the operating system is built by searching the directories subordinate to the */etc/conf/pack.d* directory.)

Here are the steps to take to create a removal script:

1. Place the removal script in the directory */usr/lib/custom.*

2. Name the removal script using this syntax:

   <productname>.rmv

   where *productname* matches the product variable listed in the permlist. (*productname* is also called the **prd** value, which is a variable listed in the permlist, and is the same value as PRODPRD in */usr/lib/site_variables*.)

3. While a shell script is not the required format for the rmv script, it is recommended because shell scripts are powerful, portable, and easy to maintain.

4. When setting permissions for the rmv script, make sure that it is executable by **custom**(ADM) on the distribution as well as in the permlist.

5. This final step is necessary only if you run either **mkcuts**(SCO) or **mkmaster**(SCO) as a standalone utility. If you plan to run **docut**(SCO) or **mkperm**(SCO), then you can skip this step.

   Add a line to the permlist for the removal script file.

A sample removal script can be found in the standard SCO UNIX System V Extended Utilities set under */usr/lib/custom/unixos.rmv*. The Extended Utilities removal script is a good example of how a single removal script can perform different actions based on which package is being removed.

Checklist

Now that you have written the scripts required for your distribution, you are ready to run the utilities described earlier in this guide. These utilities actually prepare your permlist, which includes listings for the scripts prepared here for installation. Before continuing, you may want to use the following checklist to ensure that you have completed the scripts as accurately and completely as possible:

In writing the init script, did you complete the following tasks?

- ❒ Write an init script for special installation requirements, such as relinking the kernel.
- ❒ Copy the init script into *./tmp* and name it with the syntax *init.<productname>* or *init.<pkgname>*. If your product is divided into packages, locate the init scripts subordinate to the package directory. The name must be of the form *init.PKGNAME*. If your product is not divided into packages, the name should be of the form *init.productname*, where *productname* is the abbreviated product name (also called the **prd** value, which is a variable listed in the permlist, and is the same value as PRODPRD in */usr/lib/site_variables*).
- ❒ Within the script, make sure to return a zero (0) status for successful installation and a one (1) status in case of failure.

In writing the prep script, did you complete the following tasks?

- ❒ Write a prep script for activities that must take place before installation, such as saving the users' files before updating the product.
- ❒ Place the prep script in *./tmp/perms* and name it with the syntax *prep.<productname>*.

In writing the removal script, did you complete the following tasks?

- ❒ Write a removal script for special removal requirements, such as undoing an activity performed by the init script or removing files that are not listed in the permlist. Write the script so that it returns an exit status of 0 or 1.
- ❒ Place the removal script in */usr/lib/custom* and name it with the syntax *<productname>.rmv*.

In writing each of the scripts, did you remember the following?

- ❒ Set *root* permissions to read and execute.

Chapter 5

Installing Device Drivers

Introduction

This chapter explains how to apply the SCO standard installation procedure to Installable Device Drivers (IDDs). You should be familiar with the **docut** and **custom** utilities before using this chapter; special considerations are described that you must take into account when using these utilities to install device drivers.

The information in this chapter is applicable to both SCO XENIX System V and SCO UNIX System V unless otherwise noted.

The procedure used to make an IDD **custom** installable differs from that used to make a normal application **custom** installable in one central way: the *init.idd* script can execute any number of installation scripts sequentially, waiting until all scripts are executed before relinking the kernel.

The sections about making an SCO XENIX System V or SCO UNIX System V device driver **custom** installable describe the general procedures you use to prepare a driver for **custom** installation. The section "Creating the UNIX Cutting Hierarchy" describes the files needed for the installation and where they are located. An equivalent section is provided for SCO XENIX System V. The section "Understanding the Driver Scripts" explains the various installation scripts and their purposes. The section "Examples" gives an example of each of these scripts. Use these examples, along with the information in the "Guidelines" section, to construct your scripts. Finally, the section "Compatibility with SAMI" discusses considerations for using SCO XENIX System V Semi-Automated Mass Installation.

XENIX IDD Samples

The code examples at the end of the chapter are applicable to SCO UNIX System V only. For information about SCO XENIX System V installation scripts for device drivers, examine the source code provided with your Toolkit. These files are in the */usr/lib/petkit/samples/xenix.idd* directory. A similar directory is provided for SCO UNIX System V drivers.

Making a UNIX Device Driver custom Installable

Use the following procedure to make a device driver **custom** installable:

1. Compile the program with the following syntax:

   cc -c -K -DINKERNEL *<filename>* **-o** *<Driver>*

   where **-c** creates a linkable object file, **-K** removes debugging stack probes, **-o** is the flag for an object file, and *<Driver.o>* is the name of the compiled driver. The **-D** flag with the **INKERNEL** value is required by many device driver routines.

   To ensure binary compatibility across operating systems, you may want to compile your binary with one of the flags described in the section "Compiling Compatible Binaries" in Appendix C of this guide.

2. Create a device node specification file in the */etc/conf/node.d* directory. This file is the same name as the driver's internal name.

3. Create the installation scripts. These scripts (*init.idd* and *install.drivername*) check if the driver is already configured, resolve interrupt and I/O conflicts as necessary, and add the driver to system configuration files.

4. Create a removal script (*drivername.rmv*) that can be run at a future time if you want to remove the driver.

5. Create a cutting hierarchy. Do not put device (*/dev*) files in the hierarchy. When the distribution is installed, device nodes are created for you based on the information in the system configuration files.

6. Run the **docut**(SCO) utility to create the permlist and to cut the distribution.

7. Test the installation by running **custom**.

8. Debug the driver if necessary, and then use **docut** and **custom** until the driver installation is complete.

Creating the UNIX Cutting Hierarchy

The following files are needed under the cutting hierarchy for an SCO UNIX System V IDD installation of a driver named *xx*:

Filename	Purpose
/etc/conf/pack.d/xx/driver.o	compiled driver
/etc/conf/node.d/xx	node specification file
/tmp/init.idd	generic installation script
/tmp/install.xx	driver-specific installation script
/tmp/perms/xx	permlist
/usr/lib/custom/xx.rmv	driver-specific removal script

Optional files:

Filename	Purpose
/etc/conf/pack.d/xx/space.c	local driver routine definitions that are compiled each time the operating system is built
/etc/conf/pack.d/xx/tunable.h	local driver header file definitions that relate to tuning the driver for differing hardware usage or for performance improvements
/etc/conf/pack.d/xx/stubs.c	local driver routines that are stubbed out to conform to differing installation environments, such as hardware or software changes

This cutting hierarchy is directly beneath the *root* directory.

Making a XENIX Device Driver custom Installable

Use the following procedure to make a SCO XENIX System V device driver **custom** installable:

1. Compile the program with the following syntax:

   cc -O -M3e -DM_KERNEL -DSYSINFO *<filename>* **-o** *<driver>*

   where **-O** optimizes the device driver, **-M3e** indicates that you are compiling for a 386 environment, and the two **-D** values are for compatibility with other SCO XENIX System V device drivers. The **-o** flag indicates the output file.

2. Code your installation script to make repeated calls to **configure**(C). Use **configure** to obtain a new major device number. Then **configure** is called in the installation script to check to see if the driver is already configured, resolve interrupt and I/O conflicts as necessary, add the driver to the system configuration files, install subordinate components, modify system parameters, and install the device driver. The installation script in SCO XENIX System V calls the **mknod**(CP) utility each time a device node is created. Refer to both the **configure** and **mknod** man pages for more information. The installation script is called *init.idd*. In addition, the installation script should determine if the link kit is installed and prompt the user to install if it is not. Finally, the installation script needs to add driver-specific information to the *link_xenix* file on the target computer.

3. Code a removal script (*drivername.rmv*) that can be run at a future time to remove the driver. Again, this script should call **configure** to get the major number, decide if driver is already removed, remove entries from the system configuration files, and remove all associated device nodes.

4. Create a cutting hierarchy that does not contain any device (*/dev*) files. The installation file creates these as needed when the actual installation takes place. Ensure that all the permission modes are correct for the files in the hierarchy.

5. Run the **docut**(SCO) utility to create the permlist and to cut the distribution.

6. Test the installation by running **custom**.

7. Debug the driver if necessary, and then use **docut** and **custom** until the driver installation is complete.

Creating the XENIX Cutting Hierarchy

The following files are needed under the cutting hierarchy for an SCO XENIX System V IDD installation of a driver named *xx*:

Filename	Purpose
/usr/src/conf/io/driver.o	compiled driver
/usr/src/conf/io/driver.c	driver source
/usr/src/conf/link_xenix	kernel build script
/usr/src/conf/master	master driver list
/usr/src/conf/xenixconf	switch tables
/tmp/init.idd	generic installation script
/tmp/install.xx	driver-specific installation script
/tmp/perms/xx	permlist
/usr/lib/custom/xx.rmv	driver-specific removal script

This cutting hierarchy is directly beneath the *root* directory.

Understanding the Driver Scripts

Using the **custom** utility to install drivers differs from using it to install applications in one major respect: the *init.idd* script (called */tmp/init.idd*) provides a phased installation so that multiple drivers can be installed while performing the relinking operation only once.

The *init.idd* script performs a variety of functions, including verifying the presence of the link kit, installing third-party software, relinking the kernel, and running all of the driver installation scripts located in */tmp*.

The driver-specific install scripts, located in the */tmp* directory, perform the actual driver installation.

The driver-specific removal scripts, located in the */usr/lib/custom* directory, are used to remove an installed driver with the **custom** utility.

The next several pages show the following samples:

- a permlist containing driver packages
- a UNIX *init.idd* script
- a UNIX driver installation script
- a UNIX driver removal script

Appearing after these examples are general guidelines for writing your own scripts and a checklist of required script components.

A Sample IDD Permlist

```
# Filename: ./tmp/perms/xxd
# Sample UNIX permissions list for the xxd product which includes the
# xx driver.
#
#prd=xxd
#typ=n386
#rel=1.0.0a
#set="The XX Device Driver Product"
#ser=""
#
#
# User id's:
#
uid     root    0
uid     bin     2
#
# Group id's:
#
gid     root    0
gid     bin     2
gid     sys     3
#
#
#!ALL    0       Entire XX Driver Set
#
#
PERM     F644    root/root       1       ./tmp/perms/xxd             00
PERM     F700    root/root       1       ./tmp/perms/prep.xxd        00
#
#!XXRT   0       XX Driver Run Time Files
#
XXRT    x711    bin/bin         1       ./usr/bin/runxx             00
XXRT    f700    root/root       1       ./usr/lib/custom/xxd.rmv    00
#
#!XXLK   0       XX Driver Link Kit Files
#
XXLK    F700    root/root       1       ./tmp/init.idd              00
XXLK    F700    root/root       1       ./tmp/install.xxd           00
XXLK    f644    root/sys        1       ./etc/conf/node.d/xx           00
XXLK    x644    root/sys        1       ./etc/conf/pack.d/xx/Driver.o 00
```

A Sample UNIX init.idd Script

```
:
#       Filename: ./tmp/init.idd
#       Sample UNIX init.idd Script
#
#
PATH=/bin:/usr/bin:/etc
LANG=english_us.ascii # Used when scripts are internationalized.
export PATH LANG

tmp=/tmp/idd$$          # Define a temporary file for use if necessary.
_THIRDPARTY=            # Set this variable if there is third party
                        # software to be installed. For example, some
                        # networking products use third party drivers which
                        # are installed in conjunction with their product.
# Define return values.
: ${OK=0} ${FAIL=1}

# Function definitions
########################

# Define traps for critical and non critical code.
set_trap()  {
        trap 'echo "\nInterrupted! Exiting ..."; cleanup 1' 1 2 3 15
}
unset_trap()  {
        trap '' 1 2 3 15
}

# Remove tmp files and exit with the status passed as argument.
cleanup() {
        trap '' 1 2 3 15
        [ "$tmp" ] && rm -f $tmp*
        [ -f /tmp/install.* ] && rm -f /tmp/install.*
        exit $1
}

# Print an error message.
error() {
        echo "\nError: $*" >&2
}

# Prompt for yes or no answer - returns non-zero for no.
getyn() {
        while  echo "$* (y/n) \c"
        do      read yn rest
                case $yn in
                [yY])   return $OK                      ;;
                [nN])   return $FAIL                    ;;
                *)      error "Please answer y or n"        ;;
                esac
        done
}
```

A Sample UNIX init.idd Script

```
# Although this routine is not used in this script, it is provided as
# a standard way of prompting a user for information.
# Prompt with mesg, return non-zero on q.
prompt() {
        while  echo "\n${mesg}or enter q to quit: \c"
        do     read cmd
               case $cmd in
               +x|-x) set $cmd                                    ;;
               Q|q)   return $FAIL                                ;;
               !*)    eval `expr "$cmd" : "!\(.*\)"`              ;;
               "")    # If there is an argument use it as the default
                      # else loop until 'cmd' is set
                      [ "$1" ] && {
                            cmd=$1
                            return $OK
                      }
                      : continue
                      ;;
               *)     return $OK                                  ;;
               esac
        done
}

# Set PERM variable used in linkchk().
permschk () {
        if [ -f /etc/perms/extmd ]; then
              PERM=/etc/perms/extmd
        elif [ -f /etc/perms/inst ]; then
              PERM=/etc/perms/inst
        else
              error "Cannot locate LINK packages permlist.
Needed to verify linkkit installation"
              cleanup $FAIL
        fi
}

# Test to see if link kit is installed.
linkchk()  {
        cd /
        until  fixperm -i -d LINK $PERM
        do     case $? in
               4)  echo "\nThe Link Kit is not installed."       ;;
               5)  echo "\nThe Link Kit is only partially installed." ;;
               *)  error "fixperm failed testing for Link Kit.  Exiting."
                   cleanup $FAIL  ;;
               esac

               # Not fully installed. Do so here.
               echo "\nThe link kit must be installed to run this program."
               getyn "\nDo you wish to install it now?" || {
                      # Answered no.
                      echo "\nExiting ..."
                      cleanup $FAIL
               }

               # Answered yes, so install the link kit.
               echo "\nInvoking /etc/custom\n"
               /etc/custom -o -i LINK || {
                      # Custom exited unsuccessfully.
                      error "Custom failed to install Link Kit successfully."
                      cleanup $FAIL
```

```
                }
        done
}

# Re-link new kernel.
klink() {
        # The _RELINK variable is set in the enviroment if relinking the
        # kernel is to be delayed as is the case with SAMI.
        [ "$_RELINK" ] && return

        cd /etc/conf/cf.d
        ./link_unix
}

# Install third party software.
thirdparty() {
        getyn "
Do you wish to install the third party driver or product now?" || return
        echo "\nInvoking custom ...\n"
        /etc/custom || {
                error "Custom failed installing third party software. Exiting."
                cleanup $FAIL
        }
}

# main()
########################
cd /

# Clean up and exit after signals.
set_trap

# Set PERM variable to permlist containing LINK Package.
permschk

# Check to see that the link kit is installed, since drivers cannot be
# installed without it.
linkchk

# If there is a thirdparty driver or software to be installed then the
# _THIRDPARTY variable is set to true. In that case we want to install that
# software using custom here.
[ "$_THIRDPARTY" ]  &&  thirdparty

# Run install.driver for each driver selected (sets up driver-specific info).
# A driver install script should be present in /tmp for each package the user
# selected from custom.
for i in /tmp/install.*
do
        sh $i || {
                echo "\n$i script failed. Exiting ...\n"
                cleanup $FAIL
        }
done

# Relink the kernel with the new driver information.
klink
cleanup $OK
```

A Sample UNIX Driver Installation Script

```
:
#       Filename: ./tmp/install.xxd
#       Sample UNIX install script for the xx driver.
#
PATH=/etc:/bin:/usr/bin
LANG=english_us.ascii        # Used when scripts are internationalized.
export PATH LANG

# Define return values.
: ${OK=0} ${FAIL=1} ${TRUE=0} ${FALSE=1}

# Set driver dependent variables.
name=xx
funcs="xxopen xxclose xxread xxwrite xxioctl"

# Function definitions
########################

# ---------- STANDARD ROUTINES -------- These routines are common to scripts
#                                       requiring kernel configuration.

# Define traps for critical and non critical code.
set_trap()  {
       trap 'echo "\nInterrupted! Exiting ..."; cleanup 1' 1 2 3 15
}
unset_trap()  {
       trap '' 1 2 3 15
}

# Remove tmp files and exit with the status passed as argument.
cleanup() {
       trap '' 1 2 3 15
       [ "$tmp" ] && rm -f $tmp*
       exit $1
}

# Print an error message.
error() {
       echo "\nError: $*" >&2
}

# Configure error message.
conferr()  {
       error "Configure failed to update system configuration.
Check /etc/conf/cf.d/conflog for details."
}
```

```
# ---------- CONFIGURE ROUTINES -------- These routines extract information
#                                        from and modify kernel configuration
#                                        files and parameters.

# This routine sets cf_state and the device major number if applicable.
# It returns TRUE only if the driver is fully installed.
# Possible values for cf_state are as follows.
#       NO  : Device not present in mdevice file and therefore has
#             no major number.
#       PART: Device present in mdevice file but the -Y option in the configure
#             field of the sdevice file is not set. This causes the driver to
#             be excluded from the next reconfiguration.
#       YES:  Device has a major number and the -Y option is set in
#             the sdevice file.
# Assumes we're in the /etc/conf/cf.d directory.
confchk () {
        major=`./configure -j $name` || {
                cf_state=NO
                return $FALSE
        }
        cf_state=PART
        if [ -f ../sdevice.d/$name ]
        then
                fieldval=`awk '{print $2}' ../sdevice.d/$name` || {
                        error "awk failed to read sdevice file."
                        cleanup $FAIL
                }
                [ "$fieldval" = "Y" ] && {
                        cf_state=YES
                        return $TRUE
                }
        fi
        return $FALSE
}

# Add driver according to cf_state.
# Assumes we're in the /etc/conf/cf.d directory.
confadd() {
        # cf_state set in confchk().
        case $cf_state in

        NO)     echo "\nAdding device to system configuration files ...\c"
                major=`./configure -j NEXTMAJOR`
                ./configure -c -a $funcs -m $major -h $name > conflog ||  {
                        conferr
                        return $FAIL
                } ;;
        PART)    echo "\nUpdating system configuration ...\c"
                 ./configure -c -a -m $major -Y  > conflog ||  {
                        conferr
                        return $FAIL
                } ;;
        esac
        echo "\n\nSystem configuration files have been successfully modified."
        rm -f conflog
        return $OK
}
```

A Sample UNIX Driver Installation Script

```
# ------- INSTALL ROUTINE ------- This routine checks the drivers status
#                                  and adds it to configuration files.

do_inst()  {
        curdir=`pwd`
        cd /etc/conf/cf.d
        # Check the state of the driver in system configuration files.
        confchk && {
                echo "
The $name driver is already present in system configuration files."
                return $FAIL
        }

        # Modify system configuration files to include driver.
        confadd || cleanup $FAIL

        cd $curdir
        return $OK
}

# main()
#########################

set_trap
cd /
do_inst || cleanup $FAIL
cleanup $OK
```

A Sample UNIX Removal Script

```
:
#       Filename: ./usr/lib/custom/xxd.rmv
#       This UNIX script removes the xx driver from the kernel. It is executed
#       by custom when the xxd product is removed from the system.
#
PATH=/etc:/bin:/usr/bin
LANG=english_us.ascii        # Used when scripts are internationalized.
export PATH LANG

# Define return values.
: ${OK=0} ${FAIL=1} ${TRUE=0} ${FALSE=1}

name=xx                              # Set driver name.

# Function definitions
########################

# ---------- STANDARD ROUTINES -------- These routines are common to scripts
#                                      requiring kernel configuration.

# Define traps for critical and non critical code.
set_trap()  {
        trap 'echo "\nInterrupted! Exiting ..."; cleanup 1' 1 2 3 15
}
unset_trap()  {
        trap '' 1 2 3 15
}

# Remove tmp files and exit with the status passed as argument.
cleanup() {
        trap '' 1 2 3 15
        [ "$tmp" ] && rm -f $tmp*
        exit $1
}

# Print an error message.
error() {
        echo "\nError: $*" >&2
}

# Configure error message.
conferr()  {
        error "\nConfigure failed to update system configuration.
Check /etc/conf/cf.d/conflog for details."
}
```

```
# Prompt for yes or no answer - returns non-zero for no.
getyn() {
        while  echo "$* (y/n) \c"
        do     read yn rest
               case $yn in
               [yY])   return $OK                  ;;
               [nN])   return $FAIL                ;;
               *)      error "Please answer y or n"        ;;
               esac
        done
}

# ---------- CONFIGURE ROUTINES ------- These routines extract information
#                                       from and modify kernel configuration
#                                       files and parameters.

# This routine sets cf_state and the device major number if applicable.
# It returns TRUE only if the driver is fully installed.
# Possible values for cf_state are as follows.
#       NO  : Device not present in mdevice file and therefore has
#             no major number.
#       PART: Device present in mdevice file but the -Y option in the configure
#             field of the sdevice file is not set. This causes the driver to
#             be excluded from the next reconfiguration.
#       YES:  Device has a major number and the -Y option is set in
#             the sdevice file.
# Assumes we're in the /etc/conf/cf.d directory.
confchk () {
        major=`./configure -j $name` || {
               cf_state=NO
               return $FALSE
        }
        cf_state=PART
        if [ -f ../sdevice.d/$name ]
        then
               fieldval=`awk '{print $2}' ../sdevice.d/$name` || {
                      error "awk failed to read sdevice file."
                      cleanup $FAIL
               }
               [ "$fieldval" = "Y" ] && {
                      cf_state=YES
                      return $TRUE
               }
        fi
        return $FALSE
}

# Remove driver from system configuration files.
# Assumes we're in the directory /etc/conf/cf.d .
confrm() {
        echo "\nUpdating system files to effect removal of driver ...\c"
        ./configure -c -d -m $major > conflog 2>&1 || {
               conferr
               return $FAIL
        }
        echo "\n\nSystem configuration files have been successfully modified."
        rm -f conflog
        return $OK
}
```

```
# ---------- REMOVE ROUTINE ----------     This routine checks the status of a
#                                       driver, removes it from configuration
#                                       files and relinks the kernel.

do_rm()  {
        curdir=`pwd`
        cd /etc/conf/cf.d
        confchk || {
                echo "
The $name driver is not currently linked in the kernel."
                return $OK
        }
        # Modify system configuration files to remove driver.
        confrm || return $FAIL

        getyn "\nDo you wish to relink the kernel now?" && {
                # Relink the kernel.
                ./link_unix
        }
        cd $curdir
        return $OK
}

# main()
########################

set_trap
cd /
do_rm  || cleanup $FAIL
cleanup $OK
```

Guidelines

Checklists

Complete the following actions to prepare a driver for **custom** installation:

Step	Purpose	Completed?
1	Select a unique prefix.	
2	Compile the driver source.	
3	Create the device node specification file.	
4	Create the *init.idd* file.	
5	Create the *install.driver* file.	
6	Create the removal script.	
7	Create the distribution hierarchy, making sure no device files are present.	
8	Run **docut**.	
9	Test the installation, and debug if necessary.	

Complete the following actions within your *init.idd* script:

Step	Purpose	Completed?
1	Check to see that the link kit is installed.	
2	Run all driver installation scripts in */tmp*.	
3	Relink the kernel.	
4	Install third-party software (SAMI only).	

Complete the following actions within your *install.driver* scripts:

Step	Purpose	Completed?
1	Check to see if the driver is already installed.	
2	Assign the major number with **configure**.	
3	Add the driver with **configure**.	

Packages

The files associated with various drivers in a distribution are considered separate packages within the set, and they must have separate permlist package names. Also, isolate link kit files such as *node.d*, *Driver.o*, and *space.c* by placing them in separate packages.

Configure

You should have a thorough understanding of the **configure**(ADM) command before writing driver initialization scripts, because this command provides much of the functionality needed when installing a driver. Refer to the **configure**(ADM) manual page in your operating system's *System Administrator's Reference* manual for more information. For SCO XENIX System V, **configure** is in the Commands (C) section of the **User's Reference** manual.

Device Naming Conventions

We recommend that device names follow standard conventions. Each driver must have a unique two- to four-character prefix identifying its routines (such as "hd" for hard disk). These prefixes are also used in the files associated with **configure.**

Serial device names should also follow naming conventions. For example, in "tty*nc*," *n* is the number of the board location, and *c* is an alphabetic character. Lowercase characters are used for regular ports, and the corresponding uppercase characters are used for modem control ports. For example, the first serial port on the device associated with COM2 would be named *tty2a*. If that port also has modem control, it would be named *tty2A*.

Compatibility with SAMI

A final consideration in preparing products that require relinking the kernel is compatibility with SCO XENIX System V or SCO UNIX System V Semi-Automated Mass Installation (SAMI). The SCO UNIX System V init.idd example shown previously in this chapter demonstrates the relinking of the kernel, which must be done in a shell function named **relink.** The first instruction line of the **relink** function must be a test of the _RELINK variable. If this variable is set, the function returns $OK (a zero status) immediately, and performs no action. If it is unset, the actions that follow within the function are executed normally. This allows SAMI to postpone relinking the kernel until it is appropriate for that particular configuration.

Another special variable that may be used in initialization scripts for applications that relink the kernel is _THIRDPARTY. The _THIRDPARTY variable is set if there is any third-party software to install in conjunction with the vendor's product. A **_thirdparty** function, such as the one included in the sample *init.idd* script shown previously in this chapter, should be included and called at the end of the *init.idd* script just prior to a successful exit (status $OK). The function should be called conditionally, based on the value of the _THIRDPARTY variable. SAMI checks for these specific variables for special installation considerations, and, therefore, the names must be exact.

Appendix A

Tools and Information

This appendix contains manual pages for the utilities described in this guide. The following outline shows you how these tools fit in with the information presented earlier.

Special SCO Product Engineering Toolkit (SCO) Utilities

Utility	Description
diskimage	reformats data to fit a floppy disk.
docut	creates an application distribution.
fdfit	fits file archives onto media volumes.
hocheck	compares a permlist with current and past distributions.
mkcuts	makes **custom** installable (application distribution cutting tool).
mkflops	creates disks from disk images.
mkmaster	runs previous **docut** script values.
mkperm	makes a product permission list.
pkgsize	updates size information for package in special permlist comment.
volno	updates volume number information for files.

The manual pages are usable on both the SCO XENIX System V and SCO UNIX System V operating systems.

Contents

Product Engineering Toolkit **(SCO)**

diskimage

creates file image for floppy disk

Syntax

diskimage *disk_size*

Description

diskimage copies data from standard input to standard output, formatting so that the data is not larger than *disk_size*.

docut

creates an application distribution

Syntax

docut [-c] [-e] [-p <Packages root>]

Options

-c creates a compressed *./cdist* hierarchy.

-e erases previously created data files and starts from scratch.

-p cuts a distribution that is in packages.

Description

The *docut*(SCO) utility provides an interactive method of cutting an application distribution. When *docut* completes, a *mkmaster* script is created and executed on the directory containing your distribution. Use *docut* as follows:

1. Copy all distribution files with the *tar*(C) utility into a new directory, for example, one named *./source*. (The *./misc*, *./dist* and *./cdist* directories are reserved names that cannot be used.).

2. Execute *docut* without parameters. First *mkperm -O dist* is invoked. You are prompted for the name of the directory containing your distribution. If the defaults are used, a permissions file is created in *./dist/tmp/perms* and an image of your distribution hierarchy is created using links to the *./dist* directory.

3. In the course of running *docut*, the following values are requested to create the *mkmaster* script:

   BLOCKING: blocking factor if required
   DEVICE: device name for archiving distribution
   FORMAT: complete command needed to format a volume
   VOLSIZE: size of each distribution volume

If the **-p** option to *docut* is specified, then distribution files are assumed to be organized in packages under the specified directory. For example, if **-p** is specified, then the pathnames of the files are in the form:

./directory/package-name/file-path

For example:

./source/GIZ/usr/bin/gizmo

source is the directory, GIZ is a package, and */usr/bin/gizmo* is a file that when the cut takes place is placed in the */usr/bin* directory.

docut calls *mkperm*(SCO) with each package directory and name so you are only prompted for package descriptions. For example, if a distribution has three packages named PKG1, PKG2 and PKG3 then the directory hierarchy would be set up as follows:

dirname

PKG1	PKG2	PKG3
pkg1 files	pkg2 files	pkg3 files

After setting up the distribution directory hierarchy, invoke *docut -p dirname*. An image of all packages layered onto each other is created in the *./dist* directory.

The **-c** flag is passed to *mkmaster*, which then creates a compressed hierarchy *./cdist* from which the distribution is cut. Note that the *./dist* directory is still created and used in this process.

The **-e** flag instructs *docut* to erase the *./misc/mkm.tun* file containing tunable variables inserted by *mkmaster*, and the *./misc/pf.data* file created by *mkperm* to store product data. Both of these files can be recreated by running *docut* without options. When *docut -e* is run, the following message is displayed:

```
This flag erases previous mkperm and mkmaster data files.
Are you sure you wish to do this (y/n)
```

Enter **y** for yes, or **n** for no.

Note

Because data files are retained, only invoke *docut* with desired parameters the first time it is used.

Error Messages

device `does not exist.`
The specified device for archiving the distribution does not exist or you do not have write access to the device. Specify the full pathname or ensure that permission modes are set correctly. Device pathnames can be viewed by displaying the contents of the */etc/default/tar* file.

`Error:` *name* `is not a directory`
The specified package directory cannot be found.

`Mkmaster core file /usr/lib/mkm.core is missing.`
`Cannot create mkmaster script without this file.`
The */usr/lib/mkm.core* file cannot be found. This file is contained in the Toolkit software. Reinstall the Toolkit software.

`Perms data file missing. Please run mkperm to create this file.`
While attempting to create the *mkmaster* script, the *./misc/pf.data* file could not be found. Run *docut* without options to create this file.

Files

/etc/fixperm
/usr/bin/mkperm
./misc/pf.data
./misc/mkm.tun

See Also

mkcuts(SCO), mkmaster(SCO), mkperm(SCO), tar(C)

fdfit

fits file archives onto media volumes

Syntax

fdfit [*options*]... *size packages*...

Options

-f *format*
The files are saved in a *format* archive. The default *format* is the first listed in */etc/fdformats*, and by convention is *tar*(C). A number may be appended to *format*; the meaning of that number depends on *format* but usually indicates the blocking factor. For example, a *format* of **tar20** means that *tar* is blocked at 20.

The archive formats listed in */etc/fdformats* include:

tar[*b*]
tar format, blocked *b*.

cpio[*c*][*B*]
cpio format, either binary or ASCII (**c**), and either default blocking or 5120 byte records (**B**).

ar An *ar*(CP) archive.

dd[*obs*]
dd(C) format, with an output record size of *obs* bytes.

-o *pattern*
pattern is the naming convention used for the output files. Every time "#" appears in *pattern*, the two digit number of the volume is substituted. There must be at least one "#" in *pattern*; the default is *#.fd*. Volumes are numbered starting at one ("01"); if there are any files too big to fit in on a single volume, these are listed as volume "00." Any old files following the naming *pattern* are removed.

-q
Does not complain about absolute pathnames, even if they are not recommended for *format*.

-e
Exits successfully even if some of the files in the packages could not be found.

-v
Prints, on the standard output, a table summarizing the arrangement chosen and the efficiency achieved.

-F *formatsfile*
Alternate list of *format* descriptions; the default is */etc/fdformats* .

-O *algorithm*
Specifies the default arranging *algorithm*:

n "Next file" - If the next file in the input list does not fit on the current volume, the current volume is finished and a new volume started. This preserves the order of the files but may waste volumes.

f "First fit" - The first file that fits on the current volume is chosen for that volume. Only when none of the files fit is the current volume finished and a new volume started.

b "Biggest fit" - Similar to *algorithm* **f**, except that the largest file possible is chosen.

s "Smallest fit" - Identical to *algorithm* **b**, except that the smallest file possible is chosen.

: Each package starts on a new volume.

+ After one package is finished, the next package may be started on that volume.

Note that the default *algorithm* can be overridden on a package-by-package basis. The initial default *algorithm* is "**f+**."

-c
Checks to see if each *package* fits on exactly one volume. No output files are generated unless an explicit *pattern* is given, and all *algorithms* are ignored.

-l *label*
The file named *label* (which is a pattern similar to **-o** *pattern*) is the first file on each volume. If *label* does not exist, it is created with zero length and a mode of 0444 (only read access for all). If *label* does exist, it must be a regular file; whatever size *label* has is used.

size, which must be given after all *options*, is the size of each volume in bytes. A suffix of "**k**" multiplies the number by 1024, "**b**" by 512, and "**w**" by 2.

packages are collections of files that are to be arranged as an indivisible unit. If the "**:**" *algorithm* modifier is in effect, each package starts a new volume; otherwise ("**+**") packages may share volumes. At least one *package* must be described after *size*:

file
The files in this package are listed in *file*; the package has the default *algorithm* applied to it. A *file* of - means the standard input.

-p[*algorithm*] *file*
Same as above, except that *algorithm* (if given) is used instead of the default.

-P[*algorithm*] *files ...*
Similar to **-p**, except that the named *files* are the package.

Description

fdfit tries to arrange groups ("packages") of files onto the fewest number of media volumes while avoiding splitting any file across two or more media volumes. Packages are not intermixed, and all links to a file within a package are placed on the same volume. Each volume has a capacity of *size* bytes, and is written in a specified *format*, such as *tar* (C) or *cpio* (C).

An arbitrary number of packages may be given. The packages are dealt with on a one-by-one basis, with each package being completely arranged before the next package is started. The *algorithm* used to arrange each package may be individually specified. Whether or not a package can start on the same volume that holds the end of the previous package may also be specified.

The output of *fdfit* is a series of files; the first file lists the files that belong on volume one, the second lists the files for volume two, and so on (up to a limit of 100 volumes).

Notes

Links in packages arranged by the "**n**" algorithm are usually scrambled.

No effort is made to ensure that directories occur in the arrangement before any files contained in them.

Some formats are inherently non-portable. Furthermore, some of the portable formats, such as *tar*, have slightly different overheads on different machines.

Under obscure conditions, linked files in large packages may cause infinite looping.

Format File

An *fdformats* file describes a file archiving format to *fdfit*(SCO). *fdfit* only understands distributed dictionary archives, such as *tar*(C), *cpio*(C), and *ar*(CP).

Each line in an *fdformats* file describes a separate *format*. The format is in the form:

fdformats volhdr filhdr volblk filrnd [+] [/] [type [size]]...

Empty lines, and lines beginning with "*," are ignored.

The first field, *format*, names the archive being described. *format* is an alphabetic name; no digits or non-letters are allowed.

The next four fields are decimal numbers giving the critical values:

volhdr
the overhead for each volume (floppy). This is the sum of the sizes of the volume header and trailer blocks.

filhdr
the overhead for each file (element) in the archive. This is the sum of the sizes of the element header and trailer blocks, and may be dependent on the length of the file's name. If so, a "+" should be specified in the miscellaneous fields section at the end of the line.

volblk
the volume blocking factor. The number of bytes written to each volume is always a multiple of this size.

filrnd
the file rounding boundary. Each element in the archive, with its header and trailer, is rounded up to the next such boundary.

Each of these values may have a suffixed unit of "**w**," "**b**," or "**k**," indicating multiplication of the number by 2, 512, and 1024 respectively.

In addition, each number may be prefixed by "**#**." This causes the number suffixed to *format*, as specified by the user with the **-f***format* flag to *fdfit*, to be used instead of the value given in *fdformats*. If the value in *fdformats* is suffixed with a unit, the value from **-f***format* is multipiled as indicated, ignoring any unit the user specified. But when *fdformats* does not indicate a unit, any unit indicated with **-f***format* is used.

After the four values is a miscellaneous information section. If this contains a "+," then the length of a file's name must be added to that file's *filhdr*. A "/" says that absolute pathnames are not a problem in this *format* .

The other fields in the miscellaneous section are a single character *type* optionally followed by a *size* value. The *type* indicates that a special file is correctly saved in a *format* archive:

b block special device

c character special device

d directory

n name space entry, such as semaphores and shared data regions

p named pipes (FIFO's)

& links to previously saved files

If *size* is a number (which may have a unit suffix), the *type* special file is always saved as if it were *size* bytes long. A *size* of "#" uses the actual file size obtained from *stat* (S). Otherwise, when no *size* is specified, zero (0) is assumed, the special file is completely described by the element header and trailer blocks.

Format File Notes

Unpredictable results happen when both "+" and "&" are specified.

The meaning of "/" is subjective - most archiving programs archive absolute pathnames, but the behavior on extraction is undesirable (extreme measures like *chroot* (C) must be employed).

See Also

ar(CP), cpio(C), dd(C), and tar(C)

hocheck

compares the permlist to current and past distributions

Syntax

/usr/bin/hocheck [-cd] *destination_directory permlist* [*permlist(s)*]

Options

-c clean. This option causes *hocheck* to clean out the directory of previous files. If this option is not used, then all the previous files are backed up by appending the filename with an '-'.

-d differentiate. This option creates the files *newdistfiles*, which are brand new to this cut and have somehow not been placed on the permlist(s). This is done by comparing the current *distfiles* with the previous *distfiles*. Note that this can only be done if hocheck was run on a previous distribution and the *destination_directory* still exists, and is used again. This option still works with the **-c** option.

Description

hocheck takes the permlist(s) given to it and systematically goes through them, comparing the entries it finds with the files in the distribution. Run *hocheck* from the root of the distribution you want to check. *hocheck* stores all the information it finds in a directory given by the user, *destination_directory*. It stores the following information in the following files:

devices lists all the devices given in the permlists, indicated by a *file type descriptor* of **b** or **c,** and the major and minor device numbers that are given.

distdirs lists all the directories that are in the distribution, but not on the permlists.

distfiles lists all the files found in the distribution that are not on the permlists.

empty	lists all the empty files defined on the permlists.
links	lists links found in the permlists.
newdistfiles	lists files found in the distribution, but in neither the previous distribution (as per the last search by *hocheck)* nor the permlist(s). Note that this is only created when using the **-d** option.
ootdist	lists all files in *root/boot/magic* that are not in the distribution.
ootperm	lists all files in *root/boot/magic* that are not in the permlists.
packages	lists all the packages found in the permlists and what permlists they were encountered in.
permdirs	lists all the directories in the permlists, but not in the distribution.
permfiles	lists all the file found in the permlists, but not in the distribution.

Files

/usr/bin/hocheck

Notes

If any file produced by *hocheck* is empty, then it is not be created.

See Also

mkcuts(SCO)

mkcuts

makes custom-installable distributions

Syntax

mkcuts [-cis]

Options

-c compresses the distribution files.

-i creates a disk image.

-s creates a sums list.

Description

This utility copies the files from the distribution to the output media.

The **-c** option requires access to the CDISTDIR variable, which contains the directory in which the compressed files are stored.

Only versions of *tar*(C) that have the **C** option can uncompress these files. Most XENIX and UNIX systems do not support this option. If the target system may not support this capability, do not use *mkcuts -c*.

If the CDISTDIR directory exists, then you are prompted to determine if you want to place your files in the specified directory. Responding with "y" causes the hierarchy to be compressed and new label and permlist files to be added. If you respond with "n" to the previous compression prompt, *mkcuts* does not compress the directory.

The **-i** option causes *mkcuts* to create a disk image that can then be copied to floppy disk using the *mkflops* utility. This option requires that the IMAGEDIR variable contain the name of an existing directory from which the image file directories are built. The actual directory in which the image file is stored is four levels lower than whatever name you specify. The directory names are derived from the following variables. Even though this seems complex, the *mkflops* utility accesses this path directly if *mkflops* is called without a pathname argument.

The directory structure is:

<IMAGEDIR>/<PRODPRD>/<PRODTYP>/<media-type>/<PRODREL>

The *<media-type>* directory tests the value of the DEVICE variable and then translates it into the following values:

DEVICE Value	Resulting *<media-type>*
*/dev/*48ds9*	48dsdd
*/dev/*96ds8*	96dsdd
*/dev/*96ds15*	96dshd
*/dev/*135ds9*	135dsdd
*/dev/*135ds18*	135dshd
*/dev/*rct0*	tape
*/dev/*rctmini*	mini

If a device cannot be found in the *mkcuts* translation table, "unknown" is entered. If "unknown" is undesirable, change the OTHER_MEDIA variable to a device name (basename). This name is used in the image directory name. OTHER_MEDIA is defined in the *site_variables* file.

In addition to the image file, an additional file, *mapping*, is inserted in the same directory. This file contains a single line, which is a formatted date string. If you are requesting a sums list in addition to the disk image, the sums information is also stored in the *mapping* file. When the **-i** option is selected, a message is displayed to provide the image filename and directory path.

If you did not specify the **-i** option, the following prompt is displayed:

```
C)ontinue, F)ormat, A)bort or S)kip to summing:
```

Enter **c** to *tar* the distribution files to your output media. You are prompted successively for each volume master in your distribution.

Enter **f** to format a volume before archiving files. The string entered in the FORMAT variable is executed to format your media. This variable is set in the *site_variables* file.

Enter **a** to stop processing for the current master. Abort makes it easy to skip to a specific volume if you need to recut a single volume (this is only viable as long as the permissions list and file sizes were not changed).

Enter **s** to not *tar* your distribution files to your output media. All other processing is performed, except for this aspect. Skip completes the current volume and lets you skip to the prompt for the next master in your distribution.

Choose an option, and press <Return>. Continue to insert volumes as you are prompted for them.

When all volumes are handled, **mkcuts** exits.

The *mkcuts* **-s** option creates a *sum*(C) **-r** checksum for each file in the distribution. This option depends on the SUMDIR variable being set to the name of the directory in which you want the output of the summing stored.

Files

/etc/default/petkit
/usr/lib/site_variables

See Also

docut(SCO), mkflops(SCO)

mkflops

creates floppy disks from mkcuts(SCO) output

Syntax

mkflops *[-fsrlR] [-k key] <prd> <type> <media> <rel> key*

Options

//<computer-name>
other computer access. This option followed by a computer name permits the image directory to be on another computer attached to yours with XENIX-NET. The other computer must have an */etc/default/petkit* file from which the IMAGEDIR variable is checked to find the image file.

-f format. This option formats the disks before writing to them. Default interleave is 2.

-f1 format interleave of 1. Same as **-f** option, only an interleave of 1 is used instead of 2.

-f2 format interleave of 2. Same as **-f** option.

-k *<key>* key descriptor. Use image files with only the key specified. This key is a two- or three-character string used for updates that is specified in the PRODUPD variable described in the *./usr/lib/site_variables* file.

-l local. This option causes *mkflops* to treat the local directory as the *image* directory instead of looking in the *site* file that *mkcuts* uses. To use *mkflops* over a network, work from the machine that the floppy images are stored on and then *cd* to the *$image* directory before typing *mkflops*.

-o other drive. Indicates that drive 1 is to be the output drive for cutting the distribution floppy disks.

-r release. This option causes *mkflops* to generate label templates in the *image/labels* directory as well as a *KEY srf* file. *mkflops* prompts you for a long and a short name for

the disk labels. Enter the information, which is used in a file that lists all the information for the distribution. This information is stored as a printer-ready form and is called the Standard Release Form (SRF). The "label" feature described in the prompts is unusable outside of SCO, but the information is shared with the SRF capability. After creating all of the floppies, *mkflops* asks you for more information for the Standard Release Form. If you do not know the answer to a question, press return and *mkflops* leaves that answer blank. The Standard Release Form output file is stored in the same directory as the disk image file.

-R release form only. This is just like the **-r** option, only it does not create disks, only the Standard Release Form information. However, the program still prompts you to insert a floppy disk. When this prompt occurs, simply press return and the disk drive is not activated.

-s sum. This option causes *mkflops* to compare the sums left in the *KEY*sums file. If an error is detected, you are prompted to retry writing to the disk.

<*prd*> This option and those that follow let you indicate that a specific product should be used for the distribution cut. "prd" is the same as the PRODPRD value in the *site_variables* file.

<*type*> The *type* option is the same as PRODTYPE from the *site_variables* file.

<*media*> The *media* is 48dsdd, 96dsdd, 96dshd, 135dsdd, 135dshd, or the contents of OTHER_MEDIA (from the *site_variables* file).

<*rel*> *rel* is the same as PRODREL from the *site_variables* file.

Description

The *mkflops* utility takes disk images created by *mkcuts* and creates floppy disks using these images. *mkflops* finds the image files using the IMAGEDIR variable that is set in the */etc/default/petkit* file. This variable indicates the starting directory from which the image directory hierarchy is based.

In addition, you can specify an absolute pathname for the image file directory, but *mkflops* still expects the hierarchy to be in the same format as that created by *mkcuts*.

The directory tree for *mkflops* is in this format:

<IMAGEDIR>/<PRODPRD>/<PRODTYP>/<media-type>/<PRODREL>

These variables are described in the *./usr/lib/site_variables* file. Although the *site_variables* file is not used by *mkflops* to determine the directory structure, you should examine this file to see where the image files are stored.

mkflops is capable of accessing an image directory on another computer that utilizes XENIX-NET for communications. This permits you to cut distributions when the floppy disk drive on your local machine is already in use, or when you are cutting a distribution onto floppy disks of a different type than the unit on your local machine.

mkflops has the additional capability of being able to create a printer-ready form that lists all of the information relevant to your distribution. This form is used by the SCO Quality Assurance department for evaluating both in-house and third-party software. While the information is arranged in a format specific to SCO, the listing can be used as a logging mechanism for maintaining audit trails of your distribution activities. This form is referred to as the Standard Release Form, or SRF.

Notes

mkflops uses an interleave of two as standard formatting on disks.
mkflops has a problem dealing with productions made with an empty KEY variable. Some editing of files (and filenames) in the *image* directory may be needed in this situation.

Files

/usr/bin/mkflops
/usr/lib/site_variables
/etc/default/petkit
image/KEY sums
image/KEY devs
image/KEY [0-9][0-9]

See Also

mkcuts(SCO)

mkmaster

runs previous docut(SCO) setup

Syntax

mkmaster *[-c] <machine-file>*

Option

-c change modes. This option lets you run *fixperm* on the distribution hierarchy to set permissions and ownerships to match those of the permlist. You must be logged in as *root* to use this option. *mkmaster* uses the *./usr/lib/site_variables* file to access previously defined variables.

Description

mkmaster cuts application distributions. Use of this utility is being phased out and you should use *docut*(SCO) whenever possible for future compatibility. The *docut* utility creates *mkmaster* with all the stored information from a previous run of *docut*. Running *mkmaster* simply repeats the effect of running *docut*.

mkmaster requires a directory called *./mch*. The utility searches this directory for the *machine-file* specified on the command line. *mkmaster* then sources this file to provide information specific to the host machine.

See Also

docut(SCO), fixperm(M), fixperm(ADM)

mkperm

makes a product permissions list (permlist)

Syntax

mkperm *[-p <prdvalue>] [-t <type>] [-r <rel>] [-o <output file>] [-O <dist directory>] pkgdir PKGNAME "pkgdesc"*

Options

mkperm writes a data file, which is read on subsequent execution. Specifying information on the command line supersedes previous values in the data file.

If the **-o** option is passed "-", then output goes to stdout.

The **-O** option specifies a directory from which you are archiving a distribution. Links are created from files in the permlist.

Note that the *fixperm -c* command must be run on this directory before archiving. On subsequent execution, this option can be turned off by using "-".

Description

mkperm is an interactive utility for creating a permissions list from a directory hierarchy. This utility is called by *docut*(SCO). The resultant permlist is sufficient as input for the **mkcuts** utility. The following is a list of required values that are prompted for if not specified. Refer to the *./usr/lib/site_variables* file for a listing of the defaults.

prdvalue:	abbreviated product name. Example: *pro*
setstring:	name of the full set or application. Example: *SCO Professional*
type:	target system
release:	release of the product
pkgdir:	root directory path for a given package
PKGNAME:	the name of the package associated with *pkgdir*
pkgdesc:	description of the package

Notes

fixperm creates a list of directories from primary permlists. These directories are not included in your permlist.

mkperm must be run from the *root* login.

Any files whose uid or gid name cannot be determined are inherited by root.

Files

/etc/fixperm
/usr/lib/site_variables
./misc/pf.data

See Also

docut(SCO), fixperm(M), fixperm(ADM), mkcuts(SCO)

pkgsize

updates package size information

Syntax

pkgsize [-cv] *permlists*

Options

-c checks the files against the package size information in the *permlist* file, reporting any package sizes that are incorrect.

-v For each package in the set (as indicated by the package name in the first field of the *permlist*), a verbose listing of the number of blocks, files, links, directories, and devices is displayed. The final field listed indicates whether the package information was updated.

Description

pkgsize is designed for use with *permlists* of a special, extended format recognized by *custom*. Comment fields of a particular syntax, which are ignored by fixperm, contain the total size (in blocks) of each package within a set. *pkgsize* updates the numeric size information. For example, a *permlist* size comment might be:

```
#! PRO  1020     SCO Professional Spreadsheet Package
```

pkgsize would update the numeric field based on the total size of all files marked as being part of the PRO package.

Notice that you need to be sure that the files themselves are available in the same relative position as is indicated by the relative pathname specified in the *permlist*. *pkgsize* recognizes link information, and only counts the size of linked files once.

See Also

fixperm(ADM), custom(ADM)

volno

updates volume number information for files

Syntax

volno [-ac] [-o *pattern*] [-f *types*] [-k *keyword*] *permlists*

Options

-a This option allows addition of volume number information if the volume number field was not yet included in the permlist. (The default action is to return an error during update if the volume number field is missing.)

-c This option does a check to see if any volume numbers need to be updated, and reports discrepancies without making changes.

-o *pattern*
The *pattern* is matched for selection of file lists. Note that the *pattern* must contain a "#," which is then replaced with a filename corresponding to the two-digit volume number. For example, *dist/#* indicates that the file lists are in files named *dist/01* , *dist/02* , and so on.

-f *types*
This option allows you to specify the types of files that occur on a distribution volume. This information is necessary because some archive programs do not include files of certain types.

The allowed file *types* are:

a archive

b block device

c character device

d directory

e empty

f regular

p named pipe

x executable

The default setting recognizes empty, regular, executable, and archive files.

-k *keyword*
A *keyword* may be specified, which is then prepended by *volno* onto the volume number in the permlist. This is useful if there are several packages within a single distribution set, and the volumes within a single package are to be distinguished from those in other packages.

Description

volno is designed for use with *fdfit* (SCO) to update a permlist suitable for use by *fixperm* with the appropriate volume number for each file in the package. *fdfit* (SCO) produces a file for each volume needed, containing a list of the appropriate distribution files to be included in the named volume. *volno* is then designed to use the information contained in these files to update a permlist file with the volume number for each file of a distribution.

The default *pattern* is that produced by *fdfit* (SCO).

See Also

fdfit(SCO) and fixperm(ADM).

Appendix B

Customizing SCO Open Desktop or SCO Xsight Applications

Developers of applications running under SCO Open Desktop or SCO Xsight should be aware of several special features that can enhance their product. These include running the application in its own named window, creating a specialized icon, and creating special rules for the program's icon and its data files.

Named xterm Windows

You can name the window that your program runs in by using the following syntax in your program's shell script:

xterm -T "*bigname***" -n "***littlename***"**

where *bigname* is the name that appears at the top of the xterm window and *littlename* is the name that appears on the iconified window.

Running your Program in its Own Window

If you are running an xterm window under SCO Xsight, and you want to run your application in another window, type an ampersand (&) after the program's name and press ⟨Return⟩. For example, to run SCO Professional you would type:

pro &

Creating an Icon for your Program

If you do not create a specialized icon for your program, SCO Open Desktop or SCO Xsight uses whatever icon is defined on your system to indicate an executable program.

You can use any bitmap editor to create a customized icon for your program. Pre-existing and blank icons, which you can copy and edit, are located in the */usr/include/X11/bitmaps/desktop_icons* directory. Name the icon files in the form:

<icon-name>.px

The files should be owned by *bin* and have 444 permissions modes. For more information, see the discussion of creating icons in the *SCO Xsight Configuration Guide* or in the book "Configuring the Desktop" in the *SCO Open Desktop Administrator's Guide*.

Including your Program on the Default Desktop

To include your program on the system-wide default desktop, add a line to the */usr/lib/X11/xdt/xdtinitial.xde* to indicate where the icon should appear on the default desktop; this file describes the desktop geometry. Once you place the icon here, it appears on each user's desktop, unless they have defined their own *.xde* file. You can show your users how to alter their personal files to contain your program.

Then place a shell script with the same name as your icon file in the */usr/lib/X11/xdt/defaultDesktop* directory that contains rules describing what actions should occur when the icon is selected. The files in this directory are shell scripts with 755 permissions owned by *bin*.

Altering Icon Rules

Each icon has rules that govern what happens when mouse buttons are pressed or when other file icons are dragged onto it. For example, a database program's icon may have a rules file that instructs it to execute and load any data file that ends with the suffix *.db* that is dragged on top of it.

The rules for your icon are located in the *.xdtdirinfo* file in the */usr/lib/X11/xdt/defaultDesktop* directory. The following are items typically found in an icon rule file:

- What the icon is named.
- What picture file is used for the icon.
- What happens when you double-click the mouse on it.
- What happens when you drop a file (or type of file) on it.

The *.xdtdirinfo* file also contains information about what happens when an icon is removed from the desktop, the desktop layout, and file locking.

For more information, see the discussion on altering icon rules in the *SCO Xsight Configuration Guide* or in the book "Configuring the Desktop" in the *SCO Open Desktop Administrator's Guide*.

Appendix C

Compiling Compatible Binaries

A COMPATIBLE BINARY is a program that can run on several operating system releases. With SCO XENIX System V or SCO UNIX System V, you can write one binary to run on both 286- and 386-based products. Creating a single binary for use on both systems increases the customer base for your product and simplifies updates. This section presents the various system releases and machines that your application can run on if you compile it with options for compatible binaries.

Note that for device drivers, however, a different compile process is required. See Chapter 5, "Installing Device Drivers," for information on compiling device drivers.

Table C-1 shows which flags to use with **cc**(CP) when compiling an application to make a compatible binary.

Table C-1. **Development System Compatibility**

Flag	Level	Target Systems
-compat	1	XENIX 286 System III
-xenix	2	XENIX 286 and 386 System V pre-2.3
-x2.3	3	XENIX 286 and 386 System V 2.3
-x3.2	4	SCO System V 3.2; other 386 UNIX System V
-xout	4	SCO System V 3.2; other 386 UNIX System V
-unix	4	SCO System V 3.2; other 386 UNIX System V
-coff	4	SCO System V 3.2; other 386 UNIX System V

Using the appropriate flag, you can compile a single binary to run in all of the environments in this list. A flag with Level 1, for instance, is binary compatible across all the target systems listed (Levels 1 through 4); a flag with Level 4 is compatible across the target systems listed for Level 4 only.

Here are further descriptions of some of these flags:

- The **-xenix**, **-x2.3**, or **-xout** flags generate standard XENIX files, (OMF format files). By default, the **cc** command generates files in COFF format. See the section on creating XENIX binaries in the *XENIX Development and Portability Guide* for more information.

- The **-xenix** flag, used for XENIX cross development, produces XENIX programs in OMF format. Using the XENIX libraries and include files, these resulting programs are compatible with XENIX 386 System V OS. If used in conjunction with the -M2 option, this flag is compatible with both 286 and 386 System V.

- The **-x2.3** flag is equivalent to the **-xenix** option, but it additionally includes the extended functions available with XENIX 386 release 2.3. When used in conjunction with **-M2**, this option produces XENIX 286 2.3 compatible files.

- The **-compat** flag is binary compatible across all the systems listed in Table C-1. The XENIX libraries are used with this option.

  The **-compat** flag automatically passes the **-v3, -c2,** and **-r** flags to the linker (**ld**(CP)), which results in special header information. The XENIX libraries are used with this option. The header indicates that the binary is "compatible" and can be executed on all systems listed. The **-compat** flag also defaults to the **-M0** flag, which indicates that there are no 286 instructions. (The SCO XENIX System V **cc** always defaults to using **-M0**, unless the default was changed in */etc/default/cc*.) The **-compat** option also links in both the new **brkctl**(S) library, and the proper *signal.o* library in the first segment.

The **brkctl**(S) library, introduced with SCO XENIX System V Release 2.2, emulates **brkctl**(S) on an 8086 machine using shared memory. (The **brkctl(S)** library allocates additional segments for data storage.) This is the ideal environment in which to create a compatible binary. However, it is still possible to create compatible binaries on earlier releases of SCO XENIX System V.

Here are the steps to take in creating a compatible binary:

1. Compile the program using the following syntax:

   cc [*option*] *file*

   where *option* is the name of the flag and *file* is the name of your application program.

2. If required, use the appropriate flag described in Table C-1 to create backwards compatibility for your application.

Glossary

The following terms, used in the guide, are defined here for your convenience:

BINARY. A program consisting of executable code.

COMPATIBLE BINARY. A **BINARY** containing special information that allows it to be executed in a variety of environments.

CUT. The process of moving files from hard disk to removable media, such as floppy diskettes.

DISTRIBUTION. The physical media that contains the files necessary for installation of an application. Also used to refer to the files themselves.

FIXPERM. A utility that modifies files and devices to correspond to the information found in a permissions list.

IDD (Installable Device Driver). Any product that links into the kernel.

INITIALIZATION SCRIPT. A shell script or program that the **custom**(ADM) utility executes at the end of a product installation; can be used to provide special installation requirements.

INSTALLATION. The process of transferring files from removable media to a permanently accessible location (for example, a hard disk).

LABEL. A zero-length file, distributed on a media volume, which conveys a large amount of installation information via a series of carefully named nested directories.

PACKAGE. A logical division of an application that is presented by **custom**(ADM) as a separately installed entity.

PERM. A reserved package recognized by **custom**(ADM); the permlist and any prep scrips are always included in the PERM package.

PERMLIST. A file that contains a list of all files, along with related information used by **fixperm**(ADM) and **custom**(ADM), necessary to install an application.

PREPARATION SCRIPT. A shell script or program that **custom**(ADM) executes before any other activity during installation; can be used, for example, to back up existing files before extracting new ones.

REMOVAL SCRIPT. A shell script or program that **custom**(ADM) executes before removing a package or product; can be used to provide special removal requirements.

SAMI (Semi-Automated Mass Installation). An SCO product designed to aid in the process of installing a large number of systems by requiring only a single end-user-type installation of software.

SCRIPT. A series of Bourne shell commands gathered in a single executable text file designed to perform a task or series of tasks.

SET. All packages that together make up a product.

STANDARD INSTALLATION. A set of provisions that allows a product to be **custom**-installable and that conforms to suggested conventions.

USER-CONFIGURABLE. Any item that is specifically designed to allow users to make changes to meet their needs; also referred to as system-specific.

Index

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V

X

25 June 90
507-000-001
40685

507-000-011
40685